An Evaluation of Automatic Lip-syncing Methods for Game Environments

Arien Kock[1, 2] and Jonathan Gratch[2]

[1]University of Twente, Department of Computer Science

[2]University of Southern California, Institute for Creative Technologies

ICT Technical Report No:
ICT TR 01.2005
ICT Emotions Project: 53-0820-8640

An Evaluation of Automatic Lip-syncing Methods for Game Environments [1]

Arien Kock[2, 3] and Jonathan Gratch[3]

[2]University of Twente, Department of Computer Science
[3]University of Southern California, Institute for Creative Technologies

Abstract

Lip-synching is the production of articulator motion corresponding to a given audible utterance. The Mission Rehearsal Exercise training system requires lip-synching to increase the believability of its virtual agents. In this report I document the selection, exploration, evaluation and comparison of several candidate lip-synching systems, ending with a recommendation. The evaluation focuses on the believability of articulators' expression, the foreseeable difficulty of integration into MRE's architecture, the support for facial expressions related to semantics and prosodic features as well as the scalability of each system.

[1] This work was sponsored by the U. S. Army Research, Development, and Engineering Command (RDECOM). The content does not necessarily reflect the position of the policy of the Government, and no official endorsement should be inferred.

Preface

This report is the basis for the evaluation of my practical training, which took place at the USC Institute for Creative Technologies in Marina del Rey from September 1st 2004 to December 2nd 2004.

ICT's website [2] contains a clear description of the activities it is primarily involved in: "Part of the University of Southern California, the Institute for Creative Technologies is an award-winning research center that advances the state-of-the-art in virtual reality and immersive environments. The goal of the ICT is the creation of the Experience Learning System (ELS), which provides the ability to learn through active, as opposed to passive, systems. ICT will create the ELS by harnessing creative talent from the entertainment and game development industries and leading edge research on simulation technologies such as Artificial Intelligence, Graphics and Sound. In addition to specific military training tasks, the ELS will have applications for a broad range of educational initiatives.".

To fully understand this paper it is recommended to have some knowledge of polygonal 3D graphics and animation principles.

I'd like to take the opportunity to thank the people at ICT who made my stay there a pleasant and memorable experience.

Table of Contents

1. Introduction

One of the active projects at ICT is the Mission Rehearsal Exercise (MRE). Lending from [3]: "The MRE is a virtual reality training environment…The focus of the research is on creating highly realistic and compelling face-to-face social interactions with virtual characters.". The virtual characters are able to express themselves through speech. In order to make the visual representation of these agents believable, the audible speech must be accompanied by an animated face to give the appearance of articulation. The process of animating the face is called lip-synching.

MRE is (at the time of this writing) in the process of being migrated to a new architecture. Doing so has the unfortunate consequence of losing the current lip-synching system. I was assigned the task of evaluating and comparing the performance of several systems in order to clarify the choice of a replacement system. The results were to be presented to an audience of MRE project members.

Several candidate solutions were brought to my attention. They included work done on other projects within ICT: Leaders, Flatworld and the Pighin project. MRE's new graphical subsystem is being built on the UnrealEngine 2.5, which is a videogame engine used to drive many popular game-titles [4]. Integrated into this engine is a lip-synching system called Impersonator. After initial exploration of Impersonator and the UnrealEngine 2.5 environment I discovered that it would not be necessary to look at additional systems. The choice of this scope is due to the UnrealEngine environment as well as the limited responsibility of the lip-synching system inside MRE. This is explained in the first part of the 'Candidates' chapter.

Before being able to make a comparison I needed to understand how each system worked. To be able to understand the systems I first had to become familiar with the principles of lip-synching itself. The background information contained in the next chapter attempts to clarify the basic ideas behind lip-synching that are relevant to the assignment. The 'Candidates' chapter is divided into parts that discus the details of each of the systems; how they work and what features they offer. In some instances I also briefly discuss the process I went through to better understand the systems. The 'Results' chapter contains documentation of the comparison, starting with the selection of evaluation criteria followed by the evaluation of the individual systems and ending with the comparison results. The 'Conclusions' chapter summarizes the findings of the comparison process to produce a concise recommendation.

2. Background

Lip-synching has a history founded in two-dimensional cartoons. Artists would relate the sounds of speech to a facial expression. They developed charts to help them in this process [6].

Technology has made it possible to automatically extract sounds from digitized audio and categorized into phonemes. Phonemes are described in [6] simply as: "the individual sounds that make up speech". They are in essence a set of sounds which are used to build words. Different languages have different sets of phonemes. In the Japanese language the 'S' and the 'SH' sounds are variations of the same phoneme [13], called allophones. During speech the choice of allophone is affected by the surrounding sounds, but since English speakers see 'S' and 'SH' as two distinct sounds (phonemes) and not variations of the same sound, they will not interchange them regardless of the sound's context. To go from the audible to the visual; phonemes are categorized into visual representations of the face, called 'visemes'. Not much unlike the charts used by 2D animators. The higher the number of visemes, the more precise you are able to visually represent each phoneme individually. Often differences between two phonemes are caused only by the use of vocal cords (or lack thereof), whose effects can not be seen by looking at a speaker's face. Other facial expressions are not identical but similar enough to be expressed by the same face. Mapping the phonemes down to a small number of visemes gives artists less expressions to pose. This is the main reason to map the phonemes down to a smaller number. Other reasons would be to conserve disk space, system memory and lines of code, but these do not weigh as heavy as the optimal use of man-hours. You can read more about the choice of the right number of visemes in [6].

3D lip-synching solutions animate models using morphing and blending techniques to interpolate from one viseme to the next. This way of animation produces believable animations to the untrained eye. To others it will seem mechanic and exaggerated. Humans do not make the same face for a sound each time it is uttered. The way humans articulate a sound is heavily affected by the surrounding sounds. One can make the same sound with a large number of different setups of the articulator organs. The same way the use of allophones depends on context, so does the use of the articulators. This temporal interdependence of articulation is referred to as co-articulation. The visual effects of co-articulation can be far-reaching.

In some applications it is practical to use motion capture technology to register the subtleties of facial movements, but this is expensive and time consuming if you have a large pool of audio to lip-synch. Motion capture by itself does not enable the creation of novel motion. This makes it unpractical for systems with dynamic speech content.

Sometimes a virtual face must be able to express a broad range of emotions or be able to react in dynamic ways. In these cases it is not practical to have artists create each

expression individually. The solution is to duplicate the physiological aspects of the face. Creating virtual muscles in the face and combining different levels of contraction will allow you to make almost any face imaginable. Each expression is defined as a list of values for each muscle contraction, called a muscle macro. This same parameterization of the face can be used to describe visemes as well as emotional expressions, which can easily be combined by superimposing the values in the muscle macros [7]. Some muscles affect the same part of the skin surface. If both are active, the combined influences on that surface may produce an undesired effect. Muscle-based lip-synching systems must take care that such counteracting muscles do not cause the face to take a shape that is physically impossible or interfere with more important motion.

3. Candidates

MRE has a TTS (Text-to-Speech) engine which is used for all speech output. The systems presented in this chapter are all capable of working in conjunction with this TTS engine. The audio analysis functionality of a candidate lip-synching system will not be used inside MRE. The lip-synching system is only responsible for creating the facial animations from the phoneme list generated by the TTS engine. The animation methods highlighted in the 'Background' chapter (viseme based, muscle based and motion capture) are all represented in this selection of systems. Because these systems cover the spectrum of possible solutions so well and any other systems would be equally or more difficult to integrate, I deemed it unnecessary to review additional systems.

3.1 Leaders

"The ICT Leaders Project is a research effort aimed at developing the next generation of training applications for leadership development within the US Army. The project is a collaborative effort between the Institute for Creative Technologies at the University of Southern California and Paramount Pictures." [9]

I met with Paul Carpenter, the programmer responsible for part of the lip-synching work done on the Leaders project. He demonstrated some results of his work to me. Lip-synching was not an integral part of Leaders, and completing it was not a necessity. It was made clear to me that what I was seeing was a short-lasted exploration of the possibilities to do lip-synching within the UnrealScript environment. The effort was abandoned before any substantial achievements were made.

UnrealScript is a programming language that is fully integrated into the UnrealEngine. It was created to enable everyone who owns the game to change almost all parts of the game without having access to or needing to recompile the engine itself. It is described in [12] as having the following design goals:

- “To support the major concepts of time, state, properties, and networking which traditional programming languages don't address.”
- “To provide Java-style programming simplicity, object-orientation, and compile-time error checking…The major programming concepts which UnrealScript derives from Java are: a pointer-less environment with automatic garbage collection; a simple single-inheritance class graph; strong compile-time type checking; a safe client-side execution "sandbox"; and the familiar look and feel of C/C++/Java code.”
- “To enable rich, high level programming in terms of game objects and interactions rather than bits and pixels.”

I was shown a scenario with a single lip-synching character. Its movements were being driven by skeletal animations. The UnrealEngine’s animation system allows you to restrict animations to a sub-tree of the skeletal hierarchy. An animation can be played on a channel. Any animations played on lower channels will be fully or partially masked by that animation, unless the sub-trees they are being restricted to do not overlap; there will be no interference. A channel can be assigned an alpha value which defines how much it should be blended with the animations on lower channels. One can make this alpha value change over time, consequently fading the animation on that channel in or out. This is not the only way to combine animations. When you play an animation on a specific channel you can choose to have the previous animation on that channel blend into the new one to make the transition less abrupt. For more info about UnrealEngine’s animation system refer to [12].

The code was not much more than list of animation commands with pauses in between, which indicated that it was just a hard-coded example and not a fully functional lip-synching solution. There were four versions of the lip-synching animation. An audio clip from the Leaders project was used to create a viseme schedule with the CSLU toolkit. The original viseme schedule was edited to produce different animations. One was filtered by hand, trying to include all ‘important’ information. Two were sampled at a constant rate of 5Hz and 10Hz, and the last one was sampled randomly. The animation commands used the same channel and morphed from one viseme to the next. Each of the nine visemes was a single-frame animation, since a viseme is a static pose and not a motion by itself. The duration of the morphing was constant at 0.175 seconds.

3.2 Flatworld

“The Flatworld project is a mixed reality simulation environment merging cinematic stagecraft techniques with immersive media technology. Flatworld is a joint effort between the University of Southern California's Institute for Creative Technologies (ICT) and Integrated Media Systems Center (IMSC).” [8]

I was introduced to PJ McNerney in order to talk about the lip-synching being used in Flatworld. Before the meeting I had already seen a sample animation as well as a description of how they progressed from offline to on-the-fly lip-synching. I got to see some lip-synching of pre-recorded audio as well as some dynamic input from the TTS engine.

The lip-synching uses Festival and AT&T Natural Voices TTS engine for their offline and dynamic lip-synching respectively. Flatworld's graphics are built on top of Gamebryo, a multi-platform game library. Gamebryo supports blend-shape animation, which is nothing more than interpolating in between different versions of the same mesh. All vertices will move in a linear fashion from their origin to their destination. The lip-synching uses this interpolation to animate the face, where each blend-shape is a viseme. The mapping they use projects phonemes to ten visemes. When a sequence of phonemes appears, of which each phoneme is mapped to the same viseme, the resulting identical blend-shapes are not merged into a single occurrence. The blend-shapes start blending in before their corresponding sound is played; the sound is delayed. This way the animation doesn't seem to lag behind/react to the sound.

3.3 Pighin Project

"Motion capture allows the recording of high fidelity facial motion…Editing motion capture data often involves careful key-framing by a talented animator. Motion capture by itself cannot be used for automated facial animation…Machine learning puts a new perspective on motion capture. Statistical models can be learned from training sets of high fidelity recorded data and yield novel animations that capture the details of the original motions within some interpolation space" [5]. This system was developed at ICT by Frédéric Pighin and implemented by Yong Cao.

I had read paper that describes the system before I met Yong. The meeting allowed me to ask questions about the system's functionality. I later met with Frédéric Pighin and Wen Tien, who is also involved in this project, to discuss the process needed to integrate the lip-synching system with MRE. The paper and demo content were listed online. In addition to the demos I was provided with a sample animation generated from a sound I had chosen, to enable me to make a side-by-side comparison.

Motion capture data of an actor's face was used to create a large database of motion. The motion capture process uses 109 markers on the actor's face. The original set of sentence-long utterances is segmented using phoneme boundaries. The input's audio is analyzed by Festival to generate the phoneme list. The phonemes and corresponding motions are stored as elements in a database. These entries are linked together in the same order as in their original sentences, forming a linked list. After segmentation, a clustering step is performed to merge similar motions. By merging nodes in the database the linked lists are transformed into a directed graph structure. In other motion graph structures the nodes

represent static poses and the transitions are the motion, but in this motion graph (referred to as Anime Graph) the nodes are the motion and the edges connect nodes whose motions transition smoothly when concatenated.

Once the database is created it can be queried to produce an animation that matches the input phoneme list. The query maps part of the graph to the phoneme list using as few separate graph walks as possible. The output animation will be a concatenation of more than one walk of the motion graph. The transition between these walks is not smooth, because they were not connected in the original motion data. These jumps are patched by using as much motion information from the database as possible to make the two motion nodes transition smoothly. In the best case scenario there is an edge in the motion graph that connects the same two phoneme types as the jump separates. If such an edge exists its two motion nodes are blended with the two motion segments that constitute the jump. If such an edge is not present, then the preceding motion nodes of the first graph walk and the following nodes of the second graph walk are used to create a smooth transition. In the rare case that there are no preceding or following nodes in the graph walks, then they are created using velocity information from the motion.

Intonation features are extracted from the original audio and stored in the database as well. That information is used to extract motions with very similar audio feature vectors. The original motion data is decomposed to separate speech motion from eyebrow and eyelid movement as well as emotion. The output animation that results from querying the system contains all these different components. The eyebrow and eyelid movement greatly increases the realism of the face. The markers from the motion capture process are mapped to vertices in the mesh. The rest of the vertices in the mesh that are not directly influenced by motion capture data are influenced by their surrounding points. The geometry's texture map is the result of a blending process that is meant to show skin-surface deformations that the geometry and the motion data can not express, such as a wrinkled forehead. The details of the database creation and the synthesis steps can be found in [5].

3.4 Impersonator

"Impersonator is a facial animation solution integrated with Unreal Tournament 2004. It allows you to create realistic talking characters by analyzing audio WAV files. You can use 3D Studio Max or Maya to create bone poses that determine how your character will talk. You can create poses for normal or expressive talking." [11]. Impersonator is a product licensed to Epic by OC3 Entertainment to add lip-synching functionality to Epic's UnrealEngine.

Impersonator has its own audio analysis subsystem built in. If you provide a text file in combination with the audio, you can increase the quality of the phoneme detection. Impersonator's default rig has 15 interpolation targets for speech. The collective term

‘rig’ is used to describe the number of targets for a character and how they are mapped to from phonemes. The targets are posed versions of your character’s model that are exported from your modeling software and can affect one or more bones. The artist can specify which bones a pose is restricted to. This is essential to be able to combine the lip-synching with skeletal animations being played by the UnrealEngine. Phonemes get mapped to a list of blend values, one for each target. Targets can be combined to form visemes, however, in the default rig each target is a viseme itself. The targets for speech are complemented by some additional targets for eyebrow and eyelid motion. One can modify the rig to create muscle contraction targets to move the face. A representative of OC3 Entertainment informed me of the following concerning the muscle-based targets: “Our next generation of technology will allow you to set up your content with a FACS-type system. It will be available around the end of Q1 2005…The default mapping currently maps each phoneme to a single target, but you could theoretically map each phoneme to a bunch of component muscle movements like the FACS system. The problem comes when you want to add expressions. Ideally you would want to have a different mapping for the character that uses different muscles to define how the character talks with an expression active, but you can’t.” [14].

You can generate animations using UnrealEd (the level editor that is part of the Unreal Tournament 2004 game). The animations are composed of tracks (one for each target) which have keys that describe the contribution of that track’s target to the resulting pose over time. During the generation phase the audio’s intonation data together with a random key is used to generate eyebrow and eyelid animation. Once generated you save the animations to a *.LAD (Lip-synch Animation Data) file, also referred to as a controller. These animations can then be played from within the UnrealEngine. Most of the information up to this paragraph can be found in the documentation that is part of the package in [11]. The Impersonator package includes a tool called Impersonator Studio. This tool can be used to add emotional expressions to an animation as well as changing the phoneme boundaries and types. The expressions can be either a single pose which will be blended with the rest of the targets, or it can be a completely new set of targets. These ‘emotional’ targets will be combined with the neutral targets to form new ones. The emotional expressions have their own tracks with curves just like the targets do. Any animation can have any number of blended expressions.

The first thing that struck me about this system is that it’s not real-time. All functionality to create content is only available through offline tools and not at a code level. This fact prompted me to look into bypassing the content creation pipeline of Impersonator. Although that goal turned out to be unreachable, the research I did into the inner workings of Impersonator yielded some interesting results. Namely a possible alternative solution to MRE’s lip-synching woes.

The LTF (lip-synch tweak file) contains the phoneme schedule as well as the tracks and their keys in a human-readable format. I wrote a Java program that graphically plots the

key data for all the tracks pertaining to speech. An example of the output is shown in Figure 1. Each color in the diagram is a different animation track. Since Impersonator Studio displays the expression curves as splines; I suspect that the curves on the speech tracks are splines as well. In that case the plotted curves are not entirely accurate. The real curves would be much smoother and without any sharp peaks.

The animation data is a result of the phoneme-to-targets mapping and co-articulation processing. In an attempt to recreate the animations without the co-articulation I created a program to generate tracks based on the mapping alone. The resulting animation tracks when compared to those generated by Impersonator (see Figure 1) have some notable differences. The co-articulation functionality of Impersonator has the following effects on the animations:

- fewer peaks; less visemes
- longer blend-in and blend-out times for each viseme
- smoother transitions in and out of speech

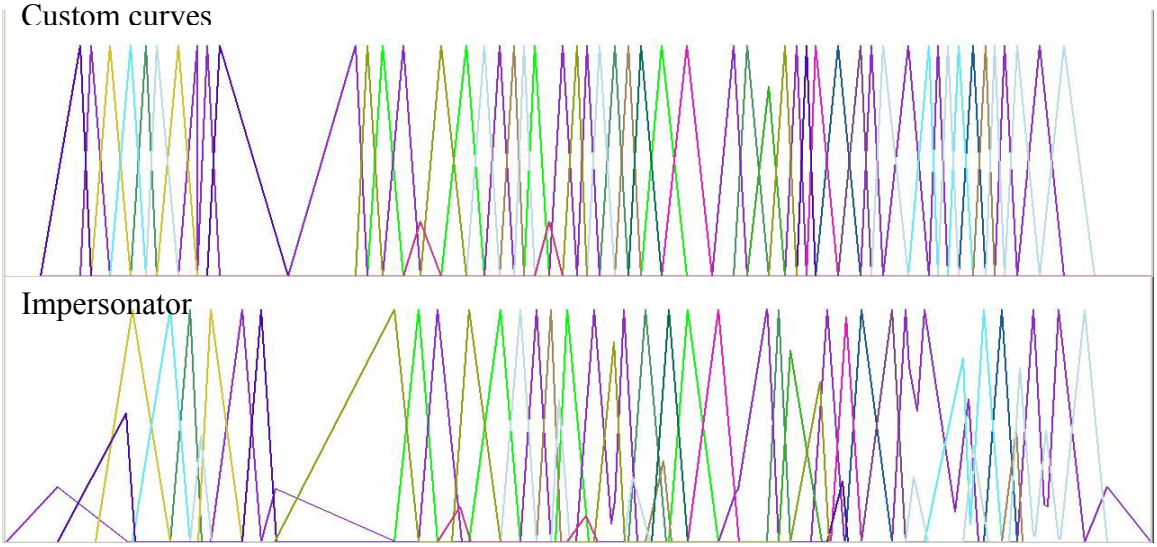

Figure 1

I experienced some problems when trying to input the generated animation tracks into the LTF and trying to view the results in UnrealEd. I believe it is likely a file format issue. To be able to make a visual comparison I changed the program that generates the animation tracks to output a list of UnrealScript animation commands. These were hard-coded animation commands specific to each utterance, similar to those in the Leaders code. The difference between these commands and the Leaders scenario is in the use of channels. The tracks generated by my program indicated how each track/viseme should behave over time, meaning that several tracks could be active concurrently. The animation commands treated each track as a channel and used the animation keys as alpha values for that track. The target poses were exported as skeletal animations and played on the channels that represent the tracks for that target. The blending of the

channels is not the same blending used by Impersonator to combine the tracks. This semantic difference did not cause any problems because the default rig never maps a phoneme to more than one target. As long as blending out of the previous viseme and blending in to the next one happens at the same time, and there are never more than two tracks with non-zero values, the resulting animation will look similar to those in Impersonator. Impersonator lets users specify which bones are affected by each target, doing so causes the animation not to interfere with any skeletal animations being played using UnrealEngine's animation system. This restriction of movement cannot be directly imitated with the skeletal animation system. Instead of a set of bones I specify a skeletal sub-tree to limit the affects of the animations. This sub-tree limits animation to only the mouth. I had to stress this point to the animators responsible for creating a new model so that the hierarchy would support such a restriction.

There are two versions of the animation command output. The first version starts blending of the next viseme at the peak of the previous viseme, which is defined as the middle of the duration of a viseme. After the peak of a viseme is reached it starts blending out and finishes at the peak of the next viseme. This cross-fading method ensures that the channel-blending is similar to the Impersonator blending. This version needs information about the current and the next viseme in order to generate the animation. The second version blends in and out of every viseme at the same speed. In contrast to the first version, no information about any viseme other than the current one is needed to play the animations, which makes it much simpler. The reason I attempted this simpler method was to make it possible to integrate it with MRE without needing to change the messages sent from the TTS subsystem. The only change on the TTS side was a new mapping. Another advantage of the second version is that no matter how short the duration of a viseme, the speed at which it blends in will be the same. This protects against really fast transitions whose erratic motion may not look natural. A difference between the animation commands and Impersonator that is important to keep in mind is the smoothness of the blending. Impersonator likely uses cubic splines to blend between curve keys, while the linear transitioning of UnrealEngine's blending function is much less smooth.

After my experiments with Impersonator and UnrealScript had finished I received a modified copy of the Impersonator library. This new version had a simple interface which could generate animation tracks given a list phoneme timing data. It was essentially a collection of functions that allowed someone to use Impersonator's co-articulation code without needing to provide an audio file. The lack of audio-analysis results in the lack of eyebrow and eyelid information. The most important feature of this interface is that it allows someone to create animations dynamically; the absence of such an interface prevented the original Impersonator to be used at all. The only limitation is that it assumes the default rig, which we can not modify. OC3 offered to add support for a custom mapping, but this would make the interface a bit more complex. I tested the new

library to ensure it worked properly and to identify any possible differences with the original Impersonator.

In order to give suitable input to the modified Impersonator library the phoneme schedule of MRE's text-to-speech engine needs to be converted to Impersonator's phoneme format. ICT's contact at OC3 Entertainment, Doug Perkowski, sent me a mapping of their (Impersonator's) phonemes to the IPA standard which included some example words (Appendix A). MRE's phoneme set for English was similar to Impersonator's which made mapping an easy task. The rest of the phonemes, including the Spanish phoneme set, were similar to the MRPA standard. Using an appendix in [1] the rest of the mapping was completed. IPA and MRPA are both phone sets (alphabets). Phones are phonetic units that are independent of language. Phonemes can and do span several phones (allophones), as with the 'S' and 'SH' example in the 'Background' chapter. These phonetic alphabets are not equal in terms of expressiveness, but each of them provides sufficient granularity to produce a meaningful mapping between the two. Since the phoneme list will not be used for speech synthesis but mapped down to visemes for facial animation, any discrepancies will not be notable. To implement this mapping some of the source code of the TTS sub-system needed to be changed.

In order to test the modified library I created a program that intercepted the messages from the TTS module. This program parses the messages and passes the phoneme timings to the Impersonator library. The output was compared to that of Impersonator Studio and appears to be identical. To get Impersonator studio to output animation tracks from an arbitrary phoneme list it must be put into a dummy LTF and loaded into Impersonator Studio, regenerating the animation tracks.

4. Results

4.1 Criteria

To make a clear comparison it is necessary to maintain a list of criteria by which candidates will be judged. In many cases the criteria which seem immediately obvious pertain to performance. Each system will be judged by the quality of the lip-synching animation. Quality will be defined as how believable the motion of the articulator organs is represented with regards to the production of sound. Since that can be argued to be a subjective metric, I will leave judging of this aspect to those responsible for the eventual choice, but still indicate what my impression of the quality is because part of my recommendation is based on it and also because I can not include the video data in this report. The time it takes for the systems to produce the animation is not as important as other criteria, since the range of acceptable speed performance is relatively broad. It would, naturally, still play a role in extreme circumstances.

The integration of the system into the current MRE architecture must be feasible. I will attempt to indicate the steps that need to be taken to integrate the systems in an attempt to measure the complexity of the overall integration process. Making the system interact with MRE requires the system's animations to be coupled to UnrealEngine. MRE's TTS output must also be mapped to a format the systems can understand. These are the main tasks involved in integration.

The believability of the entire facial animation is affected by more than jus the movement of the articulator organs. I separate two kinds of facial animation for this evaluation; animation responsible for the production of speech and the animation which aids in making it look natural. A large part of the second category has to do with revealing the internal state of the agent, because MRE's agents must be able to express emotions with their face. It would be desirable to have a lip-synching system that supports this. A facial expression pertaining to an agent's emotional state must be expressible even when the agent is not speaking. This means that the systems must be able to provide control to the emotional expressions separate from the lip-synching animation, which I will call asynchronous control. In other situations emotional expression will be bound to a certain word or part of an utterance, in these cases the expression must be synchronized with the speech motion, which I will refer to as synchronous control.

Currently MRE only has one speaking character, but there are desires to expand this in the future. Therefore scalability of the lip-synching system also comes into play. It may or may not be feasible to create the artistic content to do lip-synching for a large number of characters. Any way the systems can support such scaling will make them more attractive as long-term solutions.

To summarize, the evaluation comes down to measuring the quality, ease of integration, additional facial animations and scalability of the systems.

4.2 Evaluation

The lip-synching work done on Leaders is not a flexible system, but rather a scenario which cannot be used in MRE by itself. The approach it uses is very straightforward and in some ways less flexible than the experimentation I had done in UnrealScript. For these two reasons I am not evaluating it as a candidate solution.

Flatworld's lip-synching system is not a modular system which can be ported to MRE. The concepts implemented in Flatworld to generate animations is very similar to what Impersonator does, but without co-articulation or support for emotional expression and other facial animation. The key difference between the animations in Flatworld and Impersonator is the use of blend-shapes. Linearly interpolating between two meshes produces a different visual result than interpolating between two bone poses. The vertices in the former will move in a straight line, but may rotate around different joint positions

in the latter. The differences are not immediately obvious to the casual onlooker because the viseme meshes are not radically different, which means the vertices do not travel relatively large distances. A simple UnrealScript-based solution can produce comparable results. Even if blend-shapes improved the quality of an animation, using blend-shapes in UnrealEngine would require new animation routines to be written because UnrealEngine does not support mesh blending for skeletal meshes. The other techniques used in Flatworld are similar to those in other viseme-based systems. For all these reasons the Flatworld system will not be considered as a candidate solution.

I could only do a limited amount of integration testing, because I had no access to the source code of UnrealEngine. All integration work done or mentioned in the evaluations are estimates based on documentation of both UnrealEngine and the candidate systems.

4.2.1 Impersonator

The modified Impersonator library provides the API needed to create animation dynamically. Its performance is identical to the original library. The evaluation of the quality was done based on animations of a 3D model from the Impersonator demo content [11]. I used a phoneme schedule from the TTS engine which I mapped to the impersonator phoneme set. The resulting animation was played in Unreal. The co-articulation seems to help decrease the amount of movement, making it seem less mechanic and subtler. The slower motion of the lips and jaw attributed to a much more believable animation. Changing the rig to support muscle-based targets does not increase the quality of the poses which in turn means the animation's quality will not be affected either. This is a feature that would need additional modifications to the library by OC3 and will affect the difficulty of integration. The added freedom of expression you normally get from a muscle-based lip-synching system would not be present because the current version of Impersonator does not support multiple phoneme-to-target mappings. An option would be to write new code to handle all the mappings and use Impersonator only to play the animations. But doing this will cause the lip-synching to lose the most important thing that Impersonator has to offer, namely co-articulation. OC3 is building a system that supports fully functional muscle structures. This is more reason not to bother with any additions to the current Impersonator setup. The accuracy of the visemes plays the most important role in the quality of the lip-synching. It is important to note that Impersonator is not responsible for the precision of the visemes. The quality-affecting parts of Impersonator include the co-articulation and the smooth animation curves. The former has already been shown to increase the quality. The blending between animation keys appears to be the same smooth spline-like interpolation used in Impersonator Studio for the expression curves. Instead of simply assuming that 'smoother' means 'better', I would rather point out that no muscle movement in the human body has a constant velocity. The smooth trajectory from viseme to viseme matches the natural motion more closely.

The mapping part of the integration was done in order to test the quality of the animation with the TTS output. The mapping provided by OC3 was instrumental in this process. Impersonator uses its own animation routines but is tightly integrated with UnrealEngine already. This means that no animation porting needs to be done to get it to work. However, the controllers that are normally created offline still need to be present. The animation needs to be constructed at UnrealEngine's source level and added to the controller. The controller is associated with the proper skeletal mesh. I tested the construction of the animation, but the adding it to a controller was not. The program that tested the modified Impersonator library uses the Elvin [15] messaging system to intercept messages from the TTS engine, which had the new mapping built in. There is no audio associated with the animation, so it needs to be played separately. This could possibly cause a synchronization problem, but so far it has not been notable.

The additional animations are limited without the audio analysis step, since there is no eyebrow or eyelid movement without prosodic features. However, the animation tracks are still there, and can be filled with animation keys during the animation construction phase. If the language generator or any other part of the speech output system has knowledge of the utterance and wishes to add an eyebrow raise, it can do so. The same goes for synchronous control of emotional expressions. Making an emotional expression involves either creating one new pose or a new set of targets/visemes. Since offline specification of the curves is no longer possible, the curves need to be created dynamically by converting some form of annotation of the utterance. All control of the expressions you get in Impersonator Studio is still present, but just like the eyebrow raises it requires additional code in the system to get it to work. The changes will involve the way speech is generated, more precisely, the way emotions are associated with parts of the animation. This annotation is outside the scope of this report. Building such annotation functionality factors into the difficulty of integration, but it is something that is not limited to this system alone. Therefore I will not mention it in the evaluation of the other systems. There is no asynchronous control of the expressions relative to the lip-synch animation in Impersonator. An emotional expression can only be active during a lip-synch animation. To have responsive facial gestures that do not involve speech, a lip-synch animation needs to be created to host the expression curves. This may seem as overkill for the simpler animations like blinking. Worse than just being tedious, this animation will have to finish before you can smoothly transition to the next animation, increasing response time. In order to avoid creating a lip-synch animation every time other parts of the face need to move, one could create UnrealEngine-supported skeletal animations separate from Impersonator to do facial gestures and eye-blinks. These 'regular' animations could also be used in conjunction with lip-synch animations. The lip-synch animations will blend on top of any regular skeletal-animation since it only moves the bones relevant to speech. This method for expressions, though simpler, does not allow the expression affect the way the mouth moves, as opposed to the Impersonator expressions, which allow you to specify an entirely new viseme set. Regular animations of facial expressions/gestures and Impersonator expressions will work well together,

because the latter will fade-in and take control of part of the face as long as needed and fade-out afterwards handing control of its bones back to the regular skeletal animation. It seems an obvious solution to use regular animations for asynchronous expressions such as blinking and to use Impersonator expressions for synchronous expressions such as sad and happy emotional states. As mentioned before, a muscle model would normally help with expressions, making it possible to do combined, partial and even dynamic expressions, but the current version of Impersonator prevents us from exploiting that.

If additional characters need to have lip-synching animations, then some steps might have to be repeated. The new character will need viseme targets. The visemes need to be created from scratch, unless you use an animation tool that will remap the facial animations for you. In Figure 2 you can see the odd results of using a facial animation on a mesh it was not designed for. This viseme porting is the same problem addressed by motion retargeting, even though the viseme being retargeted is only single pose and not a complete motion. There are functions that address the special motion retargeting involved for facial animation, referred to as expression cloning. Any new model can have the original visemes ported to them by using this expression cloning. If an MRE agent's library of facial expressions and visemes grows large in size, then expression cloning becomes essential to scaling the number of agents. The size will quickly increase if one chooses to use a viseme set for each emotional expression, which according to an animation consultant at ICT is the preferred way of doing 'emotional talking'. The controllers that contain all the targets (viseme sets and other expressions) have to be created offline for every mesh. It is practical to use a 3rd party application to clone the expressions/targets instead of doing it in real-time, because the controller creation is done only once for each new character. Any tweaking done by animators to increase the quality later on will be much less labor-intensive than creating new targets.

Figure 2

4.2.2 Pighin Project

The model in this system moves as a complete and very dynamic construction. This is difficult to achieve through traditional animation methods, because hand-made skeletal animation usually does not involve such a large number of influencing factors and blend-shape animations are limited by the number of regions the face is split up in. The large number of markers captured surface stretching, which is exactly what generated animations look like; elastic stretching skin. It looks like the shape of the lips is driven by some outside force, instead of the labial muscles. The most striking example of this is the lack of inward and outward lip rotation. The way the lip moves during 'M', 'B' and 'P' sounds cannot be reproduced by the system because the capture process did not, and probably could not, track the motion of the lip's surface. Figure 3 shows that there are indeed markers on the outer ridge of the lips but none on the lip's surface. Some of the missing motion can also be attributed to the fact that after the decomposition phase a few of the signal-components were omitted. The lips should be compressed and in some cases slightly inside the mouth, but the system is only able to show a closed mouth position with the lips still in clear view. This absence of certain expected motion is the most notable shortcoming of the animation's quality. There is also no tongue movement, which is most notable during the 'N' and 'L' sounds. The accuracy of the lip-synching will improve when using the TTS phoneme schedule, because it is more accurate than a list produced by audio analysis.

Figure 3

The most difficult part of integrating this system is mapping the marker-based animation method to something supported by the UnrealEngine. One possibility would be to let the system's original code control all vertex positions in the UnrealEngine. It has been explained to me that even if this vertex-level control was possible at a source-code level, which we don't know if it is, it will probably be inefficient and slow. The marker-vertex relationship is similar enough to the bone-vertex relationship to suggest that it may be possible to replicate the effects of markers with some skeletal manipulation. The amount of influence a bone has on a vertex is called a weight. A weight of zero is equal to no influence at all, which means that no matter how much the bone moves the vertex will not be affected. The opposite is a weight of 1 which indicates that all translation and rotation of the bone will be passed onto that vertex. The collection of all the weights

bones and weights is called a smooth skin. Wen Tien told me he could output a table of influences/weights for their model without a lot of additions to the code. This table can be loaded into Maya using its scripting interface to build a smooth skinned model. From Maya it can be exported for use in the UnrealEngine. It is possible to use fewer bones than markers. The reason to use fewer bones is the lower level of detail of the MRE model, which will not be able to faithfully convey all the details of the captured motion. The unnecessary bones would only negatively affect the performance without adding to the animation quality. After meeting with Wen and Adriana we concluded that there may be another way to get a model in UnrealEngine that can be used by the motion-graph database to do lip-synching. It involves using an external tool that can remap the motion capture data to a new model using expression cloning techniques. Once the entire motion database is replicated in UnrealEngine skeletal animations the blending and time-warping needed to create novel animations can be handled by the UnrealEngine's animation functions. Blending of motion segments can be done through the same channel blending procedure I used in the UnrealScript lip-synching code. Time-warping/stretching can be achieved through the play rate parameter that defines the speed at which an animation is played, this in turn defines how long an animation takes to complete. The motion database that gets queried will contain only references to skeletal animations. The output animation will be a list of animation commands play, blend and stretch motion segments to produce the same kind of animation as the original system. The problem with these two methods is that both only drive a single model. If the expression cloning of the second method were to be done on the fly, then it would be possible to animate any model with a single motion database. However, the character model would still need a smooth skin, which should be generated in the same way as described in the first method.

Other integration issues include the texture-map blending which is not supported by the UnrealEngine. Some of the motion in the mesh triggers changes in the texture-mapping of the model. For example: a texture with wrinkles is blended onto the forehead when the eyebrows are raised. This makes the face look more natural which increases the believability of the overall animation. It may be possible to implement such a feature, but it may be a better idea to evaluate the quality of the facial animation without that texture-map blending. Since none of the current MRE meshes use this kind of blending and still look acceptable, it will likely be unnecessary to add such a feature. If texture-blending is desired in the future, then I would recommend modifying the animated textures system in UnrealEngine.

To query the system with the TTS output we need to map to the Festival phonemes that was used during segmentation. The Festival output is in the CSLU phoneme set format. This mapping needs to be created and implemented in the same way the Impersonator mapping was.

The quality of the overall facial animation was increased by the eyebrow and eyelid movement. However, just as is the case with Impersonator, the lack of any audio analysis

eliminates the animation of the upper-face. Although there is audio to be analyzed, it is synthetic and currently devoid of any meaningful prosodic elements. The decomposition of the motion capture data isolates an emotion/style signal. This can be added to an output to make the emotion appear during lip-synching. The expression affects all lip-movements in the same way and is comparable to the effects of expressions in a muscle-based facial animation system. There are currently five emotions in the database: neutral, happy, sad, frustrated and angry. Adding additional expressions would involve either new motion capture or deformation of the markers by a skilled artist. At the moment the amount of emotional expressions are controlled manually. The synchronous control of emotion should be added by those responsible for the integration of the motion with UnrealEngine's skeletal control mechanisms. To allow asynchronous control of emotion the value of the emotion, which is controlled through a slider in the original project's front-end, should be accessible to the sub-system responsible for the responsive gestures. How the effects of asynchronous expressions interact with those of synchronous control, is something which is open for discussion.

The scalability of this system depends on whether or not real-time expression cloning is used to drive the lip-synching models. If that is the case, then no new motion data will need to be generated for new geometry. Considering the 53 minutes worth of animation in the motion database, that is a very good thing. If no expression cloning is used at all, then the only way to scale the number of talking agents is to use the same facial geometry for each one. If the motion database is present as skeletal animations resulting from offline expression cloning, then the sheer size of the animation file will make it hard to have even a small number of simultaneous talking agents. Keeping only one copy of the motion database is the only way to make multiple lip-synching agents possible, but using the same geometry will look silly. That is why real-time expression cloning is essential. If the motion retargeting is in place, then the only thing needed to add a talking agent to the system is to generate a set of bones, smooth skin and the expression cloning parameters for the new model. The smooth skin needed to make the geometry move with the bones needs to be generated only once for each model, using the same method as the original motion-graph system. The smooth skin can be exported and brought into the UnrealEngine environment as described earlier in this chapter.

4.2.3 Alternative

The quality of the UnrealScript experiments is similar to that of Impersonator without the co-articulation or smooth curves. There were two different experiments: the cross-fading curves, which pre-articulated upcoming phonemes, and the constant slope curves, which had the constant blend-in time for each viseme. The cross-fading curves had the properties of not lagging behind the sound as well as not holding a pose for any amount of time. The constant slope curves could be 'stuck' in a pose for any length of time as well as blending out of a viseme before it reaches full expression. The constant slope curves could theoretically have more than one viseme being active at any one time. The

constant slope curves result in an animation with less erratic motion. When a phoneme schedule has a lot of phonemes in quick succession the first method will try to fully express each viseme while the second one will not.

Integration of the cross-fading method would require the blending in and out to occur at different times than the times provided by the phoneme schedule. This is a simple calculation that should be no problem to implement. The constant slope method is even simpler, because all it does is map a message to a function call. The new mapping at the TTS subsystem can convert phonemes directly to visemes, making it unnecessary to implement another mapping at the lip-synching side of the system. The animation process for both is already fully integrated, because it only uses UnrealEngine animation functions. The TTS output does not have to be mapped to another phoneme set; it can be directly mapped to visemes. The integration of the constant-slope method has already been performed and is the lip-synching solution currently being used by MRE.

Emotional expressions and other motion can be added to the lip-synching animation by having it play on a different sub-tree. If the expression affects part of the face which is involved in speech, it is unavoidable that the expression's sub-tree be a superset of the speech sub-tree. In that case the expression animation should be played on a lower channel than the visemes. I did some testing to add blinking to the model. There were two poses involved in blinking, one for each eye. Because the eyelids of the right eye and the left eye have the same immediate parent, it is impossible to play the two blinking animations on that parent without having them interfere with each other. The solution involved playing the animations of the eyelid bones themselves instead of a parent. This solution required me to use 2 more channels than originally intended. This problem of adding blinking teaches us that even though we have total control of any additional animations, we must carefully plan the use of channel resources and the skeletal hierarchy. Adding a new viseme set (as supported by Impersonator) instead of a single pose for a specific emotional state is achievable through tweening or using more channels. Using tweening to blend two different emotional visemes on the same channels will enable you to move between any two viseme sets over an arbitrary amount of time, but it will not let you use more than two sets in combination. Nor will it allow you to maintain a constant linear combination of the two for any amount of time, because you can not freeze the tweening. Look at the top diagram in Figure 4 for an illustration of this tweening process. The documentation of the UnrealEngine states that channel alpha-blending and animation tweening might not work well together. Since the lip-synching works by changing the alpha-values for the viseme channels, this option is unreliable. To circumvent this unreliability the tweening can be done beforehand in an animation application to create transition animations. Transition animations would replace the single-frame animations which represent a viseme. If you played the animations, they would move across different emotional visemes over time. Using an UnrealScript command to jump to any frame of the animation allows for even greater control than tweening. The middle diagram in Figure 4 shows how the transition animation fits into

the picture. You can see that the transition animation can do everything tweening can, but not the other way around. If you want to transition between any two emotions, such a transition animation may prove to be large in size as well as a lot of work to produce for every viseme. The size of the animation will depend on how many frames are used for the transition between each emotional viseme. If you want to be able to express any weighed combination of all your viseme sets, then it is not feasible to generate a transition animation to do this. The only option left is to use a different range of channels for each viseme set. With all viseme sets present and expressible at the same time, you can combine them and have them transition over time using channel alpha-blending. The bottom diagram in Figure 4 illustrates the animation channel setup for this. The white lines in the channel ranges represent the changing alpha value over time. The transition animations can produce the same results as tweening, but also offer more control in return for some extra preparation work (building the transition animations). They both use as many channels as there are visemes. The alpha-blending method uses a much larger number of channels, which is equal to the product of the number of visemes and the number of viseme sets. It offers the most control of all three options in return for using more resources and requiring more programming work. The need for the additional programming work stems from the need of some conversion calculations. The alpha value in the alpha-blending is not the same as the weight in a weighed combination, which is why the conversion is needed.

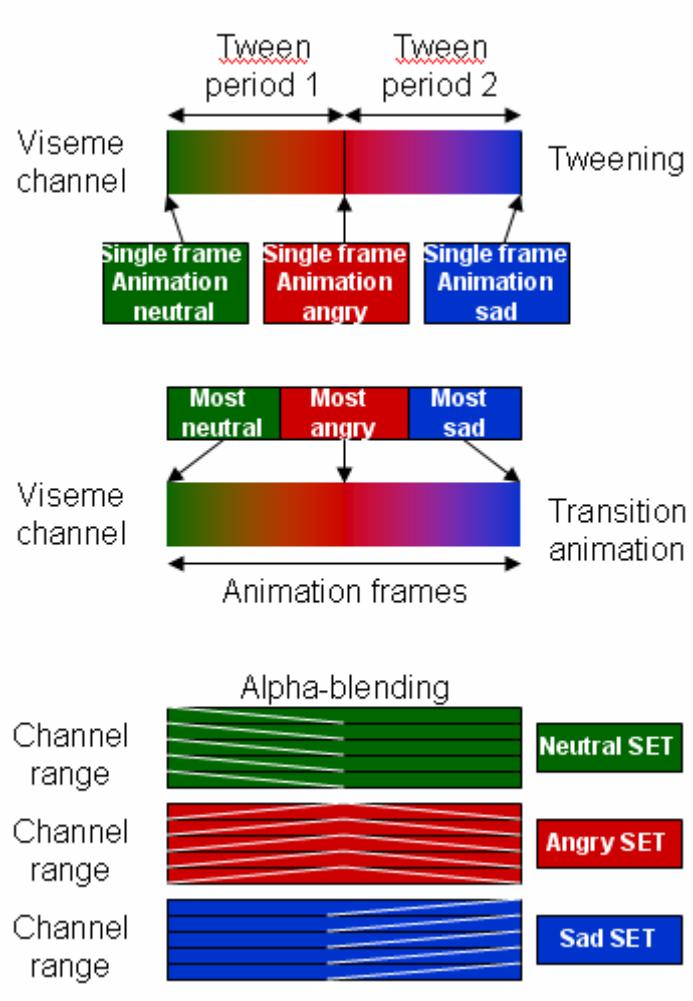

Figure 4

The scalability issues facing these alternative lip-synching solutions are identical to those of Impersonator, because they both use the static poses for visemes and expressions. To create a new lip-synching character all visemes and expressions need to be recreated either by hand or using offline expression cloning. A minor difference is that the UnrealScript solutions don't use controllers, but instead use the UnrealEngine's animation files to house all the visemes and expressions.

4.3 Compared

The definition of quality I use with regards to lip-synching is the believability; how realistic is it to assume that the facial animation could be responsible for producing the accompanied audio. This is important to keep in mind, since any constant time difference between sound and motion do not affect its quality as defined here. Any such time skew can be compensated for. The reason why I do not want to measure the quality by

comparing it to how a human would express the speech is because a lot of things factor in to the natural/human-like appearance of facial animation that do not have to do with the articulation. The motion of the face pertaining to the semantic meaning and prosodic features of the speech affect more than just the articulator organs and are not necessary to produce the sound. The criterion referred to as 'additional animations' is meant to cover this important but different category of motion.

A good way to measure this quality would be to have a side-by-side comparison of each system evaluated and rated by a large number of people. I've made a demo of such a comparison to facilitate the quality evaluation process. The systems being compared in the demo are Impersonator, the cross-fading and the constant slope alternatives. The Pighin project has not yet been integrated with UnrealEngine, which makes it impossible to compare it to the other systems in the same environment. I have video of the Pighin project system lip-synching to the same audio used in the demo. That video allows someone to see the differences in a limited manner, but the fact that one of the systems uses a different 3D model certainly influences the comparison.

The smoothness of the animation curves and the co-articulation make Impersonator's quality better than that of the cross-fading and constant slope methods. Of the two UnrealScript solutions the quality of the simpler constant slope method seems better, because in cases where phonemes occur in quick succession constant slope ensures that the virtual face does not perform unnaturally fast motion. These quickly succeeding phonemes occur more often in the TTS phoneme schedules than those from audio analysis, because the former is more precise and in general contains more elements. The quality of the Pighin project is inferior to the viseme-based approaches, because the visual form of the articulators is not expressive enough to suggest that they in fact produce sound. Although the quality would improve once it uses the TTS phoneme list as input, when its performance is compared to Impersonator using audio analysis alone it still falls behind.

All systems require a mapping to be built that converts the phonemes used by the TTS engine to either visemes or phonemes needed by the system to build the animation. The mapping for Impersonator was already created by me. The mapping for the UnrealScript lip-synching was derived from that mapping because my experiments were based on Impersonator's default rig. The mapping for the Pighin project is the only one not created, but the difficulty of creating it should be comparable to that of the Impersonator mapping.

Changes for the annotation of speech do not depend on the lip-synching system. The generation of synchronous and asynchronous commands is a feature which is also not related to the lip-synching systems. Interpreting the commands, however, is very specific to each system. Dealing with asynchronous emotion/expression commands through traditional animations is the ideal option for Impersonator and is the only option for the

two UnrealScript solutions. To use Impersonator expressions for synchronous control, the speech annotations must be converted to animation keys for the corresponding animation track. This is straightforward and by all means much simpler than writing code needed to control emotions and expressions with the Pighin lip-synching system.

Making the lip-synching systems' animations work with UnrealEngine and MRE is a much tougher task than the mapping. Of all systems the easiest to integrate is the constant-slope method, because its animations are created using UnrealScript commands and its input does not require any changes in the rest of the system with exception of the new mapping. The cross-fading method uses the same mapping, which outputs visemes. The cross-fading method needs to calculate the start, middle and end of each viseme. The calculation can be implemented at either script or source-code level. Oddly enough, writing this function is more complex than building and giving the list to Impersonator. Impersonator is the second easiest solution to integrate, making the cross-fading method the third easiest. The Pighin project was not built to do neither real-time lip-synching nor arbitrary control of emotions. The entire pipeline from phoneme schedule input to bone angles and positions needs to be built from scratch, making the Pighin system the hardest one to integrate with UnrealEngine and MRE.

Impersonator has the best support for additional facial animations, which are present in the form of expression files. The standard rig also has targets for eyebrow raises and blinks. Although the Pighin system supports emotions as well, the library of emotions is limited. The motion graph also has upper-face motion, but using it requires the construction of a feature vector without digital audio. The UnrealScript solutions support the use of asynchronous facial gestures the same way Impersonator would, through regular skeletal implementations. The script-based solutions can use the same method for synchronous expressions, but with a bit more effort can support new viseme sets just like Impersonator. The support for expressions among the script-based solutions and Impersonator is equal, although Impersonator beats them out simply because the support is built-in and requires no extra code. The Pighin system can not compete well on this topic, because extending the library of emotion is not simple and asynchronous gestures need to be animated through regular skeletal animations. Since the facial gestures need to be animated by hand, they will have to use the skeletal structure deduced from the markers, which will be hard for animators to work with.

Expanding to multiple lip-synching agents is possible with all systems, but with some constraints the Pighin system is by far the most scalable. If motion retargeting is used to play a single database of motion on any model, then the motion-graph system does not need an artist to animate or pose anything. Impersonator needs visemes and expressions for each model, as do the UnrealScript solutions. The poses can be retargeted just as with the Pighin system, but this process can not be taken completely out of human hands because the targets need to be put into controllers and animation files. The Pighin system requires a set of retargeting parameters to be established for each new model, so one

could argue that all systems require some degree of human interaction to scale. However, the size of the content is greater for an animation file and controller than a list of 10 retargeting parameters typically used for expression cloning.

5. Conclusions

Table 1 shown under this paragraph summarizes the results of the comparison by ranking the systems for each criterion.

Quality		Integration		Emotion/Expression		Scalability	
1st	Impersonator	1st	Constant slope	1st	Impersonator	1st	Pighin project
2nd	Constant slope	2nd	Impersonator	2nd	Constant slope	2nd	Impersonator
3rd	Cross-fading	3rd	Cross-fading	2nd	Cross-fading	2nd	Constant slope
4th	Pighin project	4th	Pighin project	4th	Pighin project	2nd	Cross-fading

Table 1

To get lip-synching to work as soon as possible the best option is the constant slope method. This method was implemented as a temporary solution and meets the current requirements of MRE. This gives project leaders time to decide what the final system will be. The cross-fading method seems to fall out of the picture, because it is equal in all aspects to the constant slope method with the exception of quality. This makes it an unattractive option, but the pre-articulation can possibly be added to the constant slope method to produce a better lip-synching solution. Extending this script-based solution to include co-articulation and viseme sets will cost more man-hours than integrating Impersonator, but may be worth it considering the price of purchasing Impersonator.

The Pighin project shows promise, but in its current form does not perform well enough to be considered as the permanent lip-synching solution for MRE. It should be re-evaluated if any notable improvements are made in the quality and the support for expressions.

Impersonator performs well and requires limited effort to be integrated. It seems to be the most ideal long-term solution for MRE's lip-synching requirements among the evaluated systems. In addition to its current merits, OC3 is developing a new system that uses the muscle-based approach addressed in previous chapters. Committing to using Impersonator has the advantage of being able to upgrade to that more advanced system in the future.

I recommend staying with the constant slope method until it is clear whether or not the Pighin project will improve within a reasonable timeframe. If it is not the intention to continue working on it, then the Impersonator library should be integrated as the final lip-synching solution.

References

[1] Bagshaw, P. (1994) Automatic prosodic analysis for computer aided pronunciation teaching. PhD Thesis. Edinburgh: Center for Speech Technology Research, University of Edinburgh.

[2] (2004). Welcome to the ICT. Retrieved November 1, 2004, from Institute for Creative Technologies. Web site: http://www.ict.usc.edu/includes/templates/frame_main.php?&bd=index

[3] n.d. (n.d.). Mission Rehearsal Exercise. Retrieved November 1, 2004, from Institute for Creative Technologies. Web site: http://www.ict.usc.edu/disp.php?bd=proj_mre

[4] n.d. (n.d.). Released Titles. Retrieved November 1, 2004, from Powered by Unreal Technology. Web site: http://www.unrealtechnology.com/html/powered/released.shtml

[5] Cao, Y., Faloutsos, P., Kohler, E., & Pighin, F. (2004). Real-time Speech Motion Synthesis from Recorded Motions. Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2004

[6] Lander, J. (2000). Read My Lips: Facial Animation Techniques. Retrieved November 2, 2004, from Gamasutra - The Art & Science of Making Games. Web site: http://www.gamasutra.com/features/20000406/lander_01.htm

[7] Lander, J. (2000). Flex Your Facial Animation Muscles. Retrieved November 2, 2004, from Gamasutra - The Art & Science of Making Games. Web site: http://www.gamasutra.com/features/20000414/lander_01.htm

[8] n.d. (n.d.). FlatWorld: The Mixed Reality Simulation Space. Retrieved on November 1, 2004, from Institute for Creative Technologies. Web site: http://www.ict.usc.edu/disp.php?bd=proj_flatworld

[9] n.d. (n.d.). ICT Leaders Project. Institute for Creative Technologies. Retrieved November 1, 2004, from Institute for Creative Technologies. Web site: http://www.ict.usc.edu/disp.php?bd=proj_leaders

[11] n.d. (2004). UDN - Two – LipSyncDownload. Retrieved November 1, 2004, from Unreal Developer Network. Web site: http://udn.epicgames.com/Two/LipSyncDownload

[12] Sweeney, T. (n.d.). UDN - Two – UnrealScriptReference. Retrieved November 11, 2004, from Unreal Developer Network. Web site: http://udn.epicgames.com/Two/UnrealScriptReference

[13] Knútsson, P. (2004). Phonemes. Retrieved November 11, 2004, from Phonetics I. Web site: http://www.hi.is/~peturk/KENNSLA/02/TOP/phonemes.html

[14] Perkowski, D. (November 2, 2004) OC3 Entertainment. Recipient: Arien Kock. USC Institute for Creative Technologies.

[15] n.d. (n.d). Elvin -- Content Based Messaging. Retrieved November 12, 2004, from Distributed Systems Technology Centre. Web site: http://elvin.dstc.edu.au/

Appendix A – Impersonator to IPA map

Impersonator's Phonetic Alphabet and Conversions to Standards

Phoneme	Description	Example	IPA	SAMPA (En)
IY	front closed unrounded vowel	tEE	i	i:
IH	front closed unrounded vowel, but somewhat more centralised and relaxed	Insect	ɪ	I
EH	front half open unrounded vowel	bEt	ɛ	E
EY	diphthong	plAY	ɛɪ	eI
AE	front open unrounded vowel	bAt	æ	{
AA	back open unrounded vowel	Arm	ɑ	A
AW	diphthong	abOUt	aʊ	aU
AY	diphthong	bIte	aɪ	aI
AH	back half open unrounded vowel	rUn	ʌ	V
AO	back half open rounded vowel	cAUght	ɔ	O
OY	diphthong	bOY	ɔɪ	OI
OW	diphthong	bOAt	əʊ	@U
UH	back closed rounded vowel somewhat more centralised and relaxed	pUt	ʊ	U
UW	back closed rounded vowel	sOOn	u	u:
ER	front half open unrounded vowel, but somewhat more centralised and relaxed	bIRd	ɜ	3
AX	schwa	About	ə	@
S	voiceless alveolar fricative	Sea	s	s
SH	voiceless postalveolar fricative	SHe	ʃ	S
Z	voiced alveolar fricative	Zone	z	z
ZH	voiced postalveolar fricative	pleaSure	ʒ	Z
F	voiceless labiodental fricative	Fat	f	f
TH	voiceless dental fricative	THin	θ	T
V	voiced labiodental fricative	Van	v	v
DH	voiced dental fricative	THen	ð	D
M	bilabial nasal	Man	m	m
N	alveolar nasal	No	n	n
NG	palatal nasal	siNG	ŋ	N
L	alveolar lateral	Law	l	l

R	retroflexed alveolar approximant	Red	ɻ	r
W	rounded back semivowel	War	w	w
Y	unrounded front semivowel	Yes	j	j
HH	voiceless glottal fricative	Hay	h	h
B	voiced bilabial stop	Bee	b	b
D	voiced alveolar or dental stop	Day	d	d
JH	voiced postalveolar affricate	JuDGE	dʒ	dZ
G	voiced velar stop	Game	g	g
P	voiceless bilabial stop	Pet	p	p
T	voiceless alveolar or dental stop	Test	t	t
K	voiceless velar stop	Key	k	k
CH	voiceless postalveolar affricate	CHance	tʃ	tS
SIL	silence	(silence)		