# Development of Text-to-Audiovisual Speech Synthesis to Support Interactive Language Learning on a Mobile Device

Wai-Kim Leung, Ka-Wa Yuen, Ka-Ho Wong and Helen Meng

Human-Computer Communications Laboratory
Department of Systems Engineering and Engineering Management
The Chinese University of Hong Kong
Hong Kong SAR, China
E-mail: {wkleung, kwyuen, khwong, hmmeng}@cuhk.edu.hk

***Abstract* — We have developed distributed text-to-audiovisual-speech synthesizer (TTAVS) to support interactivity in computer-aided pronunciation training (CAPT) on a mobile platform. The TTAVS serves to generate audiovisual corrective feedback based on detected mispronunciations from the second language learner's speech. Our approach encodes key visemes in SVG format that are compressed by GZIP and transmitted to the client, where the browser can perform real-time morphing to render the visual speech. We have also developed a TTAVS animation player that can play the audio and visual speech synchronously while enabling user controls in play/pause/resume. Evaluation shows that this newly proposed approach, vis-à-vis our original approach that involves generation of an Ogg video on the server-side which is streamed to the client, achieves a significant reduction (66%) in average size of the output files that are transmitted from the server to the client, reduction of (83%) in client waiting times, as well as preserve the quality of the image.**

***Index Terms*—** visual speech synthesizer, language learning, computer aided-pronunciation training system (CAPT)

## I. INTRODUCTION

The number of people learning a second language (L2) is increasing at a very fast pace around the world. It was estimated that in 2010 there would be 2 billion English learners worldwide and the proportion in Asia alone will exceed the number of native speakers [1]. The acute shortage of qualified English teachers calls for computer-aided pronunciation (CAPT) technologies, which present the advantage of round-the-clock, personalized and private instruction for L2 learning. Learning is a cognitive process and involves both perceptual and productive training. Perceptual training requires that the learner perceives the information communicated to them – in the case of CAPT, the information or media conveyed must strive to enhance the learner's perception of the deviances between the canonical (i.e. correct) pronunciation and the learner's incorrect pronunciation. Productive training aims to automatically analyze the learner's pronunciation to detect possible errors from the speech signal. Hence, CAPT is an artificially "cognitive" system [2] that first detects mispronunciations using automatic speech recognition technologies [3] and subsequently synthesizes corrective feedback to facilitate learning. The feedback should ideally be multimodal, akin to human-human communication, The traditional "listen and repeat" approach may not suitable for all learners, falling short when the learner fails to fully perceive the differences between the canonical and erroneous pronunciations. As such, the learners may only guess at the articulatory movements needed to generate the correct sounds, which is not an effective method of learning. This motivates us to develop technologies to effectively support CAPT and related inter-cognitive and multimodal communication [2]

Aside from developing ASR technologies for mispronunciation detection, we have also been developing speech synthesis technologies for corrective feedback generation. In 2004, Hazan et al. [4] proved that audiovisual perceptual training gives a positive effect on perception and production of consonants by Japanese learners of English. In this regard, we believe that it is important for the visualization of speech to be multimodal – involving not only the audio modality in conventional (audio-only) text-to-speech (TTS) synthesis, but also the visual modality to show the articulatory movements in producing different speech sounds. In other words, audio visual signals have been used to convey corrective feedback to the learner in pronunciation training.

Furthermore, since usage of smart handheld devices is growing at a phenomenal rate (e.g. from 1.03 billion smartphones in use in 2012 to a predicted 2 billion by 2015), we have begun to explore the provision of CAPT support on mobile platforms. This presents challenges to text-to-audiovisual speech (TTAVS) synthesis because of the need to support interactivity in a mobile (lightweight) platform, in face of the synthesis of media that involves resource-hungry processing. This paper presents our work in the development of a platform to support TTAVS for L2 learning in mobile applications.

978-1-4799-1546-0/13/$31.00 ©2013 IEEE

## II. PREVIOUS WORK

### A. Baseline Technologies with Ogg generation

We have previously developed a basic, server-based approach for TTAVS [6,7]. The output phoneme sequence of a text-to-speech synthesizer (TTS) is fed into an articulation processor (AP) that generates the corresponding visemes (i.e. visual phonemes) as output. More specifically, the AP covers articulatory components such as the upper lip, lower lip, tongue, velum, etc. Each phoneme is associated with a pair of visemes and each viseme is composed of various configurations of the articulatory components (though some articulatory components may be unspecified). A viseme is treated as a "key frame" of visual speech. We also devised an intermediate frame generator to synthesize by interpolation the unspecified articulatory components between the corresponding specifications from neighboring key frames. This interpolation method is also used to synthesize the intermediate frames between completely specified key frames. We have also added on a voicing and airflow processor (VAP) that inserts visualization of correct voicing, airflow from aspiration and plosives, as well as nasalization. All these are converted to bitmap and passed to a video encoder to compress the bitmap frames into video in Ogg Theora format (www.theora.org). This synthesis procedure is used to generate both the midsagittal and frontal views of the articulatory movements for any English free-text input. The two views together aim to help learners perceive the articulatory movements for production of correct pronunciations. We integrated both the audio and video modalities by encoding and compression and output the animation by Ogg video streaming. This text-to-audiovisual speech synthesizer has been incorporated into our home-grown CAPT system (named Enunciate) for correct feedback generation [8]. However, this baseline approach is computationally heavy, especially due to bitmap conversion and compression for each video frame. The file size of the output also tends to be too large. The computational load and file size are of concern in mobile speech applications. This motivates us to conduct research to address these problems.

### B. System Architecture of the Ogg Video Streaming Approach Approach

Figure 1 depicts the overall system architecture of the Ogg video streaming approach described in the previous subsection. The light-weight client sends an HTTP request with the input English text for synthesis to the server and all subsequent processing is conducted on the server side, until the output video file is streamed to the client for playback. Processing on the server side starts with audio speech synthesis using the open-source Flite [9] engine which generates the audio and its corresponding phoneme sequence that has a time duration for every segment. This phoneme sequence forms the input for visual speech synthesis, which consists of the following components in processing:

- *The Articulator Processor* (AP) extracts the relevant rules from the *Allophonic Rules Database*, such as the assimilation rule (/s sh/ → /sh/) and applies them to the phoneme sequence. The AP also extracts the viseme components (e.g. tongue, jaw, etc.) corresponding to the input phonemes from the *Viseme Component Database*. The viseme components include both the midsaggital and frontal views of the articulators in SVG format. Correspondence between the extracted viseme components and their time duration is also registered.
- The *Intermediate Viseme Generator* (IVG) receives the output of the AP and marks each viseme image as a *key frame*. The IVG then performs linear interpolation (or morphing) between adjacent key frames with a sufficient number of frames covering the time interval in-between. Each intermediate viseme is converted from SVG (Scalable Vector Graphics) to bitmap format.
- The *Voicing and Airflow Indicator Processor* (VAIP) is responsible for adding voicing or airflow indicator onto the frames. For example, a voicing indicator will be added to the corresponding frame when /v/ is pronounced. The VAIP outputs a sequence of bitmap frames that show the articulator motions with voicing and airflow at appropriate times.
- The *Video Generator* (VG) receives the sequence of bitmap frames from the *VAIP* and concatenates them frames sequentially to generate the video of the animation. This, together with the audio previously generated, are encoded and compressed by Xuggle [10] (the wrapper of FFMPEG [11] in Java) to generate the audiovisual animation in Ogg format.

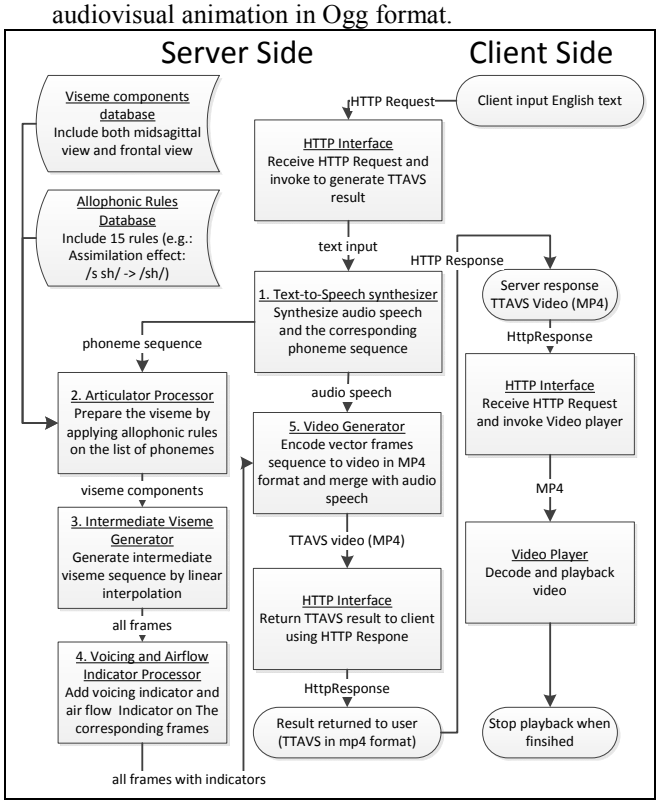


Fig. 1. System Architecture of Text-to-Audiovisual Speech Synthesis using the Ogg video streaming approach

When the Ogg video is streamed to the client, a standard video player or an HTML5-enabled browser will be able to play the animation file. Details of this process is described in [6,7]

### C. Bottlenecks

The *Ogg video streaming approach* has low requirements for the client and most clients can play the animation. However, computation is heavy on the server-side. In particular, the *Intermediate Viseme Generator* needs to generate the full setoff intermediate frames and convert each video frame from SVG to bitmap format. On average, a 10-second video with 30 frames per second (fps) will require that 300 frames be generated and converted. Thereafter, the process of compression and encoding into Ogg format by the *Video Generator* is also a computationally heavy and time consuming process. The typical size of an animation file for an utterance is 450KB and the average client waiting time (which includes the processes of video generation and streaming over Wi-Fi) is 20 seconds. This is not sufficient fast to provide an interactive user experience. In order to support CAPT in a mobile application, we need to reduce the animation file-size and the client waiting time. We propose a new approach in the next section.

## III. Approach using SVG and Client-side Morphing

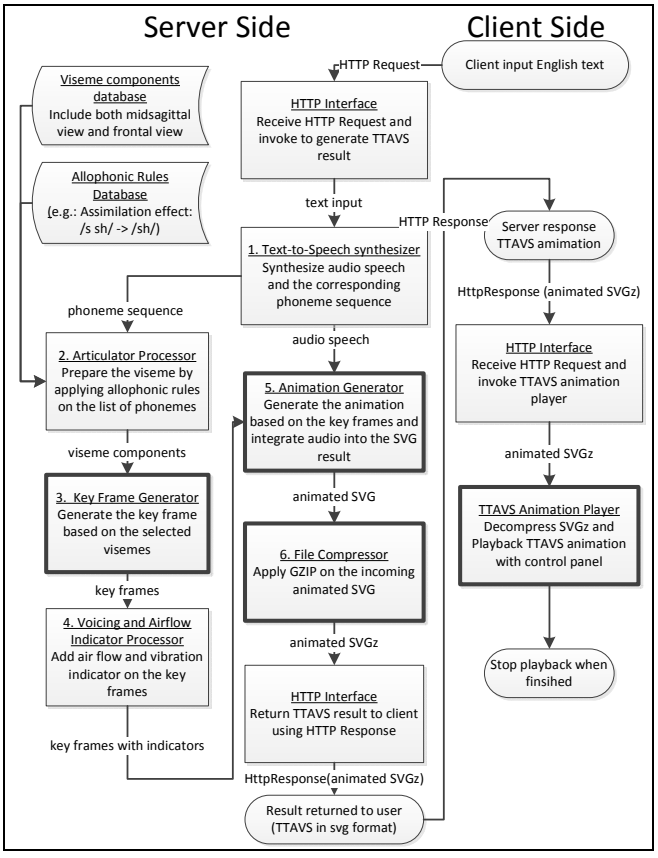


Fig. 2. System Architecture of the new SVG and Client-side Morphing Approach (Bolded boxes are modified processes)

We propose a new approach using *SVG with client-side morphing* to reduce the animation file size and client waiting time. SVG (Scalable Vector Graphics) [12] is an open standard, XML-based vector image for 2D graphics and animation which is supported by the latest, major web browsers on desktop and mobile devices. The output of our TTAVS synthesis only contains the key frames of the visual channel, together with the audio channel. All the intermediate frames between key-frames will be generated on the fly during the playback. Recall that in the *Ogg video streaming approach*, the bottlenecks are in the *Intermediate Viseme Generator* and *Video Generator*. These two processes are modified in our new approach to avoid computational heavy processes such as image format conversion and video compression. Furthermore, we add the *File Compressor* and *TTAVS Animation Player* to compress the output file and allow user to control the animation respectively. Figure 2 shows the system architecture that implements the SVG with client-side morphing approach. Modified components are shown with bolded lines and are elaborated below.

### A. Server-side Modification

*A.1 Key frame Generator* – this replaces the Intermediate Viseme Generator, and accepts viseme components from the Articulator Processor and combines the them into key frames. The key frames from the targeted viseme for each time segment. Intermediate frames between key frames are no longer generated. The generated key frames are kept in SVG format instead of converted to bitmap format to save computation. An example of a key frame is shown in Figure 3. The frame is composed of several articulators, each of which is expressed as a <path> element. This is used to define a path, such as the boxed section in Figure 3 which traces the shape of the tongue (i.e. an articulator). Table 1 further explains the attribute d used to represent a single articulator. By combining all the paths for all the articulators, a complete viseme image is generated.

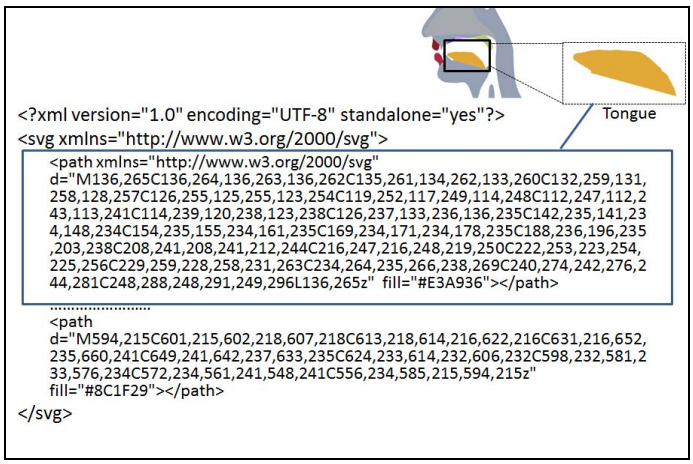


Fig. 3. Code fragment of single key frame of animation in SVG format. The boxed section represents the shape of tongue.

| Attribute | Usage |
|---|---|
| d="M136,265C136,264……..265" | Represents the actual path information. The English letters are commands for the path data and the numbers are coordinates. For example, *Mx,y* means that the starting point is *(x,y)*. *Cx1,y1,x2,y2,x,y* refers to drawing a cubic Bézier curve from the current position to *(x,y)* with two control points *(x1,y1)* and *(x2,y2)*. *Z* means drawing a straight line to the starting point to close the path. |
| fill="#E3A936" | The color "#E3A936" fills the area encompassed by the path |

Table 1. Attribute *d* used in the <path> element which represents an articulator.

*A.2 Animation Generator* – This replaces the *Video Generator* in the *Ogg video streaming approach*. It accepts all key frames (in SVG format) generated previously and combines them into one animated SVG file. The audio information, together with the syncrhonization information, is added to the SVG file simultaneously. Figure 4 illustrates an SVG file, where a list of <animate> elements are added into each <path> element to animate the corresponding articulator. The <animate> element animates its parent's value over the specified time. The boldfaced section in Figure 4 illustrates an animation that involves the articulator changing its shape. Table 2 further lists the attributes required for animation of that articulator.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<svg xmlns="http://www.w3.org/2000/svg">
      <path xmlns="http://www.w3.org/2000/svg"
   d="M136,265C136,264,136,263,136,262C135,261,134,262,133,260C132,259,
   131,258,128,257C126,.....L136,265z" fill="#E3A936">
        <animate xmlns="http://www.w3.org/2000/svg"
          repeatCount="1" attributeName="d"
          begin="0" dur="0.307s"
           to="M136,265C136,264,136,263,136,262
            C135,261,134,262,133,260.....L136,265z"fill="freeze"/>
        <animate> ..... </animate>
        .....
   </path>
  .....
</svg>
```

Fig. 4. Code fragment of animated SVG. The section in bold animates an articulator changing its shape within the first 0.307s.

| Attribute | Usage |
|---|---|
| repeatCount=1 | The animation can only be played once |
| attributeName="d" | The attribute *d* in the parent element <path> will be animated. |
| begin="0" | Animation starts at t = 0s |
| dur="0.307s" | The duration of animation is 0.307s |
| to="M136,265C….z" | The target value of the attribute indict by *attributeName* |
| fill="freeze" | The animation effect is frozen at the end time of animation |

Table 2. Attributes used in the <animate> element to represent the movements of an articulator.

Hence the overall specification of the boldfaced fragment in Figure 4 is to animate the shape of the articulator defined by the initial path towards the target shape of *"M136,265C136,264,136,263,136,262C135,261,134,262,133,260……..L136,265z"* starting at time 0 seconds with a duration of 0.307 seconds. This animation is performed only once and the endpoint of the animation will "freeze".

Audio information is added into the SVG file in HTML5 audio format. Since SVG and HTML5 are in different XML namespace, the <foreignObject> element is used to enable the inclusion of a foreign XML namespace. The *src* attribute indicates the URL of audio in different formats such as .wav and .mp3. The browser will automatically choose the most appropriate audio source. This is illustrated in the code fragment in Figure 5.

This implementation as shown in Figure 5 also enables the user to control the playback of the animation. A dummy <path> element with an <animate> element is added to the SVG, i.e. <animate begin="indefinite" dur="0s" id="videoChannel"/>. This dummy path creates an empty animation with id videoChannel where the begin time is indefinite (i.e. unspecified) and hence will not start until the use issues the call. Also the animation has zero duration. Thereafter, the begin times of all the <path> elements with <animate> elements are set to "videoChannel.end" which means all the animations will only be started after the animation with id videoChannel is complete. In the <audio> element, we add the attributes autoplay="true" and oncanplay="this.pause()"– the objective is to cache the audio and hold off the playback until it is invoked. Invocation of the start function for playback is done by providing a play button for the user in the TTAVS Animation Player, which will be described in Section III B. Hence the code fragment in Figure 5 shows the complete procedure of TTAVS animation file in SVG format.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<svg xmlns="http://www.w3.org/2000/svg">
  <path><animate begin="indefinite" dur="0s" id="videoChannel"/></path>
  <path d=" M136,265C136,264,136,263,1,.....L136,265z " fill="# E3A936 ">
     <animate repeatCount="1" attributeName="d" begin="videoChannel.end" dur="2s" to="
     M136,265C136,264,136,263,136,2627.....L136,265z"fill="freeze "/>
  </path>
  .....
<foreignObject>
  <audio id="audioChannel" autoplay="true" oncanplay="this.pause();"
xmlns="http://www.w3.org/1999/xhtml">
    <source src="university.wav" />
    <source src="university.mp3" />
    </audio>
  </foreignObject>
</svg>
```

Fig. 5. Code fragment of the complete TTAVS animation file in SVG format.

*A.3 File Compressor* – this module compresses the SVG file using the standard GZIP algorithm and output an SVGZ (Gzipped SVG) file for sending to the client. Such lossless data compression is very effective for SVG file formats since it is based on XML and contains many repeated text fragments. Gzipped content, including SVGZ, are supported by general browsers such as Chrome, Firefox, Safari and Internet Explorer.

*B. Client-side Modification*

*TTAVS Animation Player* – This module is needed to handle the received SVGZ file (which is not a video file as in the Ogg video streaming approach). This module is written in JavaScript and requires the WebKit [13] browser engine.

As the SVG file only contains the key frames of animation, morphing will be performed by WebKit to render all the intermediate frames during playback according to the values in <animate> elements. Since the interpolation mode is not specifed in the <animate> elements, the linear interpolation mode will be used by default. The player has three buttons named as Play, Pause and Resume to support the functions of play, pause and resume respectively. Synchronization of the audio and visual channels require simulataneously calling two indiviual functions to play/pause/resume the audio and visual channels respectively. Figure 6 shows the code fragements of the 3 functions, relating to HTML5 audio and SVG animation.

```
<script>
  //play animation (play button)
  function playAnimation() {
    document.getElementById("videoChannel").beginElement();
    document.getElementById("audioChannel").play();
  }

  //pause animation (pause button)
  function pauseAnimation() {
    document.getElementById("videoChannel").pauseElement();
    document.getElementById("audioChannel").pause();
  }

  //resume animation (resume button)
  function resumeAnimation() {
    document.getElementById("videoChannel").unpauseElement();
    document.getElementById("audioChannel").play();
  }
</script>
```

Fig. 6. Code fragment of the play/pause/resume buttons to play/pause/resume the audio and visual channels synchronously in TTAVS.

## IV. Experiments and Results

### A. Client Waiting Time and File Size

To compare the performance between the original Ogg video streaming approach (see Section II) and the new approach of SVG with client-side morning (see Section III), we composed 142 short English utterances for synthesis and measured the average output file size and the client waiting time. The latter includes time for server processing and network transmission over Wi-Fi. The test was repeated twice – where the first run included synthesis of the visual channel only, and the second run included both audio and visual channels. Results of the first test are shown in Table 3. The average SVGZ file size is only 94% of the average Ogg video file size. This is because SVG is an XML-based format containing many repeated text fragments which is very suitable for GZIP compression. Since the processes of intermediate frame generation and conversion, video encoding and compression are no longer performed on the server side, the new approach is able to reduce client waiting time by 90% on average.

| **Average** | **Ogg video streaming** | **SVG with client-side morphing (reduction)** |
|---|---|---|
| File size (KB) | 419.7 | 24 (94%) |
| Client waiting time (s) | 17.8 | 1.7 (90%) |

Table 3. Average file size and client waiting time using the original (Ogg video streaming) and new (SVG with client-side morphing) approaches. Only the visual channel is considered.

In the second test, both audio and visual channels are generated to reflect actual TTAVS usage. Since the audio data is binary, the lossless compression rate is low. Nevertheless, we achieved 66% file size reduction with the new approach. Inclusion of the audio increased the average client waiting time by 3.3 seconds per utterance and hence the new approach achieves a reduction of 83%. The results are shown in Table 4.

### B. Video Quality

Since the Ogg video uses lossy video compression, image quality of our original approach suffers degradation. However, the new approach of SVG with client-side morphing preserves the image quality. Figure 9 shows the screenshots of the animations generated by both approaches.

| **Average** | **Ogg video streaming** | **SVG with client-side morphing (reduction)** |
|---|---|---|
| File size (KB) | 444.5 | 150.5 (66%) |
| Client waiting times (s) | 19.5 | 3.3 (83%) |

Table 4. Average file sizes and client waiting times using the two aaproaches. Both audio and visual channels are included.

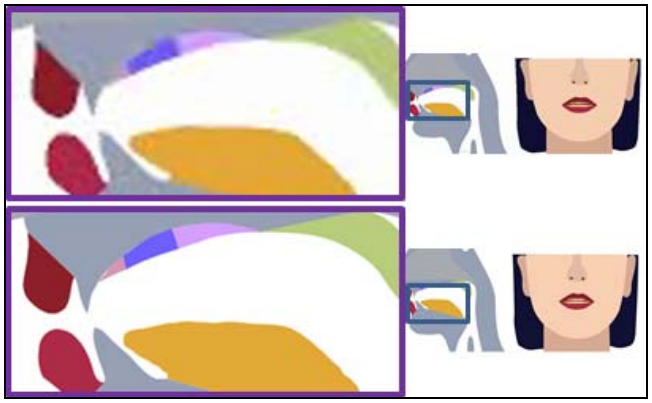

Fig. 7. Screenshot of visual speech animation by using Ogg video streaming approach(top) and SVG client morphing approach(below)

## V. Conclusions and Future Work

In this paper we present the design of a text-to-audiovisual (TTAVS) speech synthesis system that supports interactivity on mobile applications relating to computer-aided pronunciation training (CAPT). Our baseline TTAVS technology includes audio speech synthesis together with visual speech synthesis using viseme (i.e. visual phoneme) generation and interpolation across the duration of a

phoneme segment. The original TTAVS system design is based on Ogg video streaming and places most of the computation on the server-side to generate a video file which is streamed to a light-weight client. However, the generated audiovisual speech has relatively large file sizes and also long client waiting times. These are not conducive to interactivity for mobile applications. Our new proposed approach is based on SVG with client-side morphing. Major visemes are encoded as key frames of visual speech, which are generated on the server-side and encoded in SVG format. The GZIP algorithm is applied to compress the key frames to SVGZ format for transmission to the client. Most browsers on the client side can extract the key frames from the SVG file and render the animation by morphing the key frames by linear interpolation in real time. The new approach reduces the average file size of synthesis outputs by 66% to approximately 150KB per utterance. The average client waiting time is also reduced by 83% to 3.3 seconds each utterance. The video suffers minimal loss in image quality. In the near future, we will integrate our TTAVS synthesizer into the CAPT system for mobile application.

## ACKNOWLEDGMENT

The work is partially supported by Innovation and Technology Fund, Hong Kong SAR government (ITS/163/12) and CUHK Teaching Development Grant. We also thank the anonymous reviewers for their helpful comments.

## REFERENCES

[1] B. B. Kachru, “Asian Englishes: Beyond the Canon.” Hong Kong University Press, 2005.

[2] G. Sallai, “The Cradle of Cognitive Infocommunications,” Acta Polytechnica Hungarica, vol. 9, no. 1, pp. 171–181, 2012.

[3] H. Meng, W. K. Lo, A. M. Harrison, P. Lee, K. H. Wong, W. K. Leung and F. Meng, "Development of Automatic Speech Recognition and Synthesis Technologies to Support Chinese Learners of English: The CUHK Experience," Proceedings of the 2nd Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA), December 2010.

[4] V. Hazan, A. Sennema, M. Iba, A. Faulkner, “Effect of audiovisual perceptual training on the perception and production of consonants by Japanese learners of English,” Speech Communication, Volume 47, Issue 3, November 2005, pp. 360-378, ISSN 0167-6393.

[5] Numer of Smartphones around the World, http://finance.yahoo.com/news/number-smartphones-around-world-top-122000896.html

[6] K. H. Wong; W. K. Leung; W. K. Lo; H. Meng, "Development of an articulatory visual-speech synthesizer to support language learning," Proceedings of the International Symposium on Chinese Spoken Language Processing (ISCSLP), 2010.

[7] K. H. Wong; W. K. Lo; H. Meng, "Allophonic variations in visual speech synthesis for corrective feedback in CAPT," Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2011.

[8] K. W. Yuen; W. K. Leung; P. Liu; K. H. Wong; X. Qian; W. K. Lo; H. Meng, "Enunciate: An Internet-accessible computer-aided pronunciation training system and related user evaluations," Meeting of the Oriental Chapter of the International Committee for the Coordination and Standardization of Speech Databases and Assessment Techniques, 2011.

[9] Flite, http://www.festvox.org/flite/

[10] Xuggle, http://www.xuggle.com/

[11] FFMPEG, http://www.ffmpeg.org/

[12] SVG, http://www.w3.org/Graphics/SVG/

[13] WebKit, http://www.webkit.org/