A Practical Model for Live Speech-Driven Lip-Sync

Li Wei and Zhigang Deng ▪ *University of Houston*

A simple, efficient, yet practical phoneme-based approach to generating realistic speech animation in real time based on live speech input starts with decomposing lower-face movements and ends with applying motion blending. Experiments and comparisons demonstrate the realism of this synthesized speech animation.

The signal-processing and speech-understanding communities have proposed several approaches to generate speech animation based on live acoustic speech input. For example, based on a real-time recognized phoneme sequence, researchers use simple linear smoothing functions to produce corresponding speech animation.[1,2] Other approaches train statistical models (such as neural networks) to encode the mapping between acoustic speech features and facial movements.[3,4] These approaches have demonstrated their real-time runtime efficiency on an off-the-shelf computer, but their performance is highly speaker-dependent because of the individual-specific nature of the chosen acoustic speech features. Furthermore, the visual realism of these approaches is insufficient, so they're less suitable for graphics and animation applications.

Live speech-driven lip-sync involves several challenges. First, additional technical challenges are involved compared with prerecorded speech, where expensive global optimization techniques can help find the most plausible speech motion corresponding to novel spoken or typed input. In contrast, it's extremely difficult, if not impossible, to directly apply such global optimization techniques to live speech-driven lip-sync applications because the forthcoming (unavailable yet) speech content can't be exploited during the synthesis process. Second, live speech-driven lip-sync algorithms must be highly efficient to ensure real-time speed on an off-the-shelf computer, whereas offline speech animation synthesis algorithms don't need to meet such tight time constraints. Compared with forced phoneme alignment for prerecorded speech, this last challenge comes from the low accuracy of state-of-the-art live speech phoneme recognition systems (such as the Julius system [http://julius.sourceforge.jp] and the HTK toolkit [http://htk.eng.cam.ac.uk]).

To quantify the phoneme recognition accuracy between the prerecorded and live speech cases, we randomly selected 10 prerecorded sentences and extracted their phoneme sequences using the Julius system, first to do forced phoneme-alignment on the clips (called *offline phoneme alignment*) and then as a real-time phoneme recognition engine. By simulating the same prerecorded speech clip as live speech, the system generated phoneme output sequentially while the speech was being fed into it. Then, by taking the offline phoneme alignment results as the ground truth, we were able to compute the accuracies of the live speech phoneme recognition in our experiment. As Figure 1 illustrates, the live speech phoneme recognition accuracy of the same Julius system varies from 45 to 80 percent. Further empirical analysis didn't show any patterns of incorrectly recognized phonemes (that is, the phonemes often recognized incorrectly in live speech), implying that to produce satisfactory live speech-driven animation results, any phoneme-based algorithm must take the relatively low phoneme recognition accuracy (for live speech) into design consideration. Moreover, that algorithm should be able to perform certain self-corrections at runtime because some phonemes could be incorrectly recognized and input into the algorithm in a less predictable manner.

Published by the IEEE Computer Society

Inspired by these research challenges, we propose a practical phoneme-based approach for live speech-driven lip-sync. Besides generating realistic speech animation in real time, our phoneme-based approach can straightforwardly handle speech input from different speakers, which is one of the major advantages of phoneme-based approaches over acoustic speech feature-driven approaches (see the "Related Work in Speech Animation Synthesis" sidebar).[3,4] Specifically, we introduce an efficient, simple algorithm to compute each motion segment's priority and select the plausible segment based on the phoneme information that's sequentially recognized at runtime. Compared with existing lip-sync approaches, the main advantages of our method are its efficiency, simplicity, and capability of handling live speech input in real time.

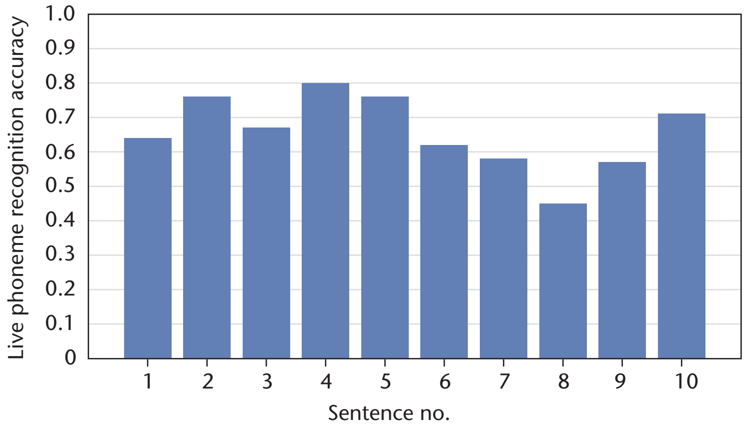

Figure 1. The Julius system. Its live speech phoneme recognition accuracy varied from 45 to 80 percent.

Data Acquisition and Preprocessing

We acquired a training facial motion dataset for this work by using an optical motion capture system and attaching more than 100 markers to the face of a female native English speaker. We eliminated 3D rigid head motion by using a singular value decomposition (SVD) based statistical shape analysis method.[5] The subject was guided to speak a phoneme-balanced corpus consisting of 166 sentences with neutral expression; the obtained dataset contains 72,871 motion frames (about 10 minutes of recording, with 120 frames per second). Phoneme labels and durations were automatically extracted from the simultaneously recorded speech data.

As Figure 2a illustrates, we use 39 markers in the lower face region in this work, which results in a 117-dimensional feature vector for each motion frame. We apply principal component analysis (PCA) to reduce the dimension of the motion feature vectors, which allows us to obtain a compact representation by only retaining a small number of principal components. We keep the five most significant principal components to cover 96.6 percent of the motion dataset's variance. All the other processing steps described here are performed in parallel in each of the retained five principal component spaces.

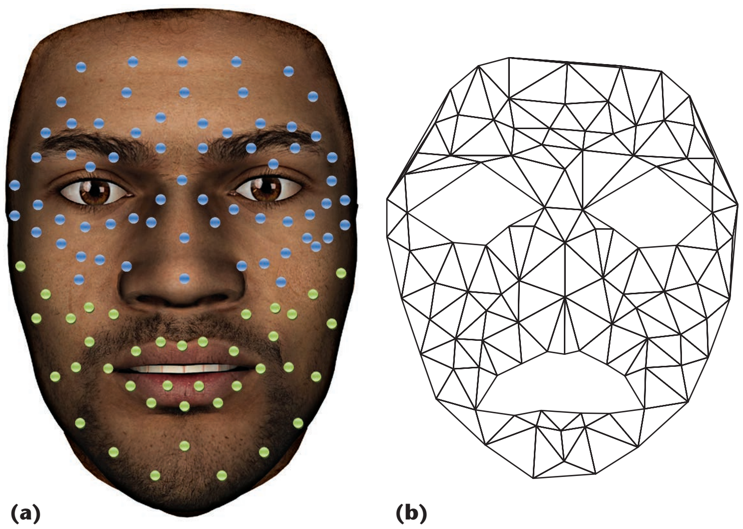

Figure 2. Facial motion dataset. (a) Among the 102 markers, we used the 39 green markers in this work. (b) Illustration of the average face markers.

Motion Segmentation

For each recorded sentence, we segment its motion sequence based on its phoneme alignment and extract a motion segment for each phoneme occurrence. For the motion blending that occurs later, we keep an overlapping region between two neighboring motion segments. We set the length of the overlapping region to one frame (see Figure 3).

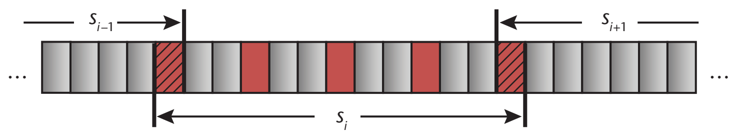

Figure 3. Motion segmentation. The grids represent motion frames, where s_i denotes the motion segment corresponding to phoneme p_i. When we segment a motion sequence based on its phoneme timing information, we keep one overlapping frame (grids with slashes filled) between two neighboring segments. We evenly sample five frames (grids with red color) to represent a motion segment.

Motion Segment Normalization

Because a phoneme's duration is typically short (in our dataset, the average phoneme duration is 109 milliseconds), we could use a small number of evenly sampled frames to represent the original motion. We downsample a motion segment by evenly selecting five representative frames (see

Related Work in Speech Animation Synthesis

The essential part of visual speech animation synthesis is determining how to model the speech co-articulation effect. Conventional viseme-driven approaches need users to first carefully design a set of *visemes* (or a set of static mouth shapes that represent different phonemes) and then employ interpolation functions[1] or co-articulation rules to compute in-between frames for speech animation synthesis.[2] However, a fixed phoneme-viseme mapping scheme is often insufficient to model the co-articulation phenomenon in human speech production. As such, the resulting speech animations often lack variance and realistic articulation.

Instead of interpolating a set of predesigned visemes, one category of data-driven approaches generates speech animations by optimally selecting and concatenating motion units from a precollected database based on various cost functions.[3–7] The second category of data-driven speech animation approaches learns statistical models from data.[8,9] All these data-driven approaches have demonstrated noticeable successes for offline prerecorded speech animation synthesis. Unfortunately, they can't be straightforwardly extended for live speech-driven lip-sync. The main reason is that they typically utilize some form of global optimization technique to synthesize the most optimal speech-synchronized facial motion. In contrast, in live speech-driven lip-sync, forthcoming speech (or phoneme) information is unavailable. Thus, directly applying such global optimization-based algorithms would be technically infeasible.

Several algorithms have been proposed to generate live speech-driven speech animations. Besides predesigned phoneme-viseme mapping,[10,11] various statistical models (such as neural networks,[12] nearest-neighbor search,[13] and others) have been trained to learn audio-visual mapping for this purpose. The common disadvantages of these approaches is that the acoustics features used for training the statistical audio-to-visual mapping models are highly speaker-specific, as pointed out by Sarah Taylor and her colleagues.[6]

References

1. M.M. Cohen and D.W. Massaro, "Modeling Coarticulation in Synthetic Visual Speech," *Models and Techniques in Computer Animation*, Springer, 1993, pp. 139–156.
2. A. Wang, M. Emmi, and P. Faloutsos, "Assembling an Expressive Facial Animation System," *Proc. SIGGRAPH Symp. Video Games*, 2007, pp. 21–26.
3. C. Bregler, M. Covell, and M. Slaney, "Video Rewrite: Driving Visual Speech with Audio," *Proc. SIGGRAPH 97*, 1997, pp. 353–360.
4. Y. Cao et al., "Expressive Speech-Driven Facial Animation," *ACM Trans. Graphics*, vol. 24, no. 4, 2005, pp. 1283–1302.
5. Z. Deng and U. Neumann, "eFASE: Expressive Facial Animation Synthesis and Editing with Phoneme-Level Controls," *Proc. ACM SIGGGRAPH/Eurographics Symp. Computer Animation*, 2006, pp. 251–259.
6. S.L. Taylor, "Dynamic Units of Visual Speech," *Proc. 11th ACM SIGGRAPH/Eurographics Conf. Computer Animation*, 2012, pp. 275–284.
7. X. Ma and Z. Deng, "A Statistical Quality Model for Data-Driven Speech Animation," *IEEE Trans. Visualization and Computer Graphics*, vol. 18, no. 11, 2012, pp. 1915–1927.
8. M. Brand, "Voice Puppetry," *Proc. 26th Annual Conf. Computer Graphics and Interactive Techniques*, 1999, pp. 21–28.
9. T. Ezzat, G. Geiger, and T. Poggio, "Trainable Video-Realistic Speech Animation," *ACM Trans. Graphics*, vol. 21, no. 3, 2002, pp. 388–398.
10. B. Li et al., "Real-Time Speech Driven Talking Avatar," *J. Tinghua Univ. Science and Technology*, vol. 51, no. 9, 2011, pp. 1180–1186.
11. Y. Xu et al., "A Practical and Configurable Lip Sync Method for Games," *Proc. Motion in Games* (MIG), 2013. pp. 109–118.
12. P. Hong, Z. Wen, and T.S. Huang, "Real-Time Speech-Driven Face Animation with Expressions Using Neural Networks," *IEEE Trans. Neural Networks*, vol. 13, no. 4, 2002, pp. 916–927.
13. R. Gutierrez-Osuna et al., "Speech-Driven Facial Animation with Realistic Dynamics," *IEEE Trans. Multimedia*, vol. 7, no. 1, 2005, pp. 33–42.

Figure 3); we represent each motion segment s_i as a vector v_i that concatenates the PCA coefficients of the five sampled frames.

Clustering

We apply a K-means clustering algorithm to the motion vector dataset $\{v_i: i = 1 \ldots n\}$, where n is the total number of motion vectors, and then use Euclidean distance to compute the distance between two motion vectors—we call the center of each obtained motion cluster an *AnimPho*. Next, we obtain a set of AnimPhos $\{a_i: i = 1 \ldots m\}$, where m is the total number of AnimPhos, and a_i is the ith AnimPho.

After this clustering step, we essentially convert the motion dataset to a set of AnimPho sequences. Then, we further precompute the following two data structures:

- AnimPho transition table, T, is an $m \times m$ table, where m is the number of AnimPhos. $T(a_i, a_j) = 1$ if the AnimPho transition $<a_i, a_j>$ exists in the motion dataset; otherwise, $T(a_i, a_j) = 0$. In this article, we say two AnimPhos a_i and a_j are connected if and only if $T(a_i, a_j) = 1$. We also define $T(a_i) = \{a_j | T(a_i, a_j) = 1\}$ as the set of AnimPhos connected to a_i.
- Phoneme-AnimPho mapping table, L, is an $h \times m$

table, where h is the number of phonemes used (we used 43 English phonemes in this work). $L(p_i, a_j) = 1$ if the phoneme label of a_j is phoneme p_i in the dataset; otherwise, $L(p_i, a_j) = 0$. Similarly, we define $L(p_i) = \{a_j | L(p_i, a_j) = 1\}$ as the set of AnimPhos that have the phoneme label p_i.

Live Speech-Driven Lip-Sync

In the work described here, two steps generate lip motion based on live speech input: AnimPho selection and motion blending.

AnimPho Selection

Assuming phoneme p_t arrives at the lip-sync system at time t, our algorithm needs to find its corresponding AnimPho, a_t, to animate p_t. A naive solution would be to take the AnimPho set, $\phi(p_t) = L(p_t) \wedge T(a_{t-1})$, which is the set of AnimPhos that have the phoneme label p_t and have a natural connection with a_{t-1}, and then choose one of them as a_t by minimizing a cost function (for example, the smoothness energy defined in Equation 1). If $\phi(p_t)$ is empty (called a *miss* in this article), then we reassign all the AnimPhos (regardless of their phoneme labels) as $\phi(p_t)$:

$$a_t = \underset{\eta \in \phi(p_t)}{\arg\min}\, \mathrm{dist}\left(E(a_{t-1}), B(\eta)\right), \tag{1}$$

where $\mathrm{dist}(E(a_{t-1}), B(\eta))$ returns the Euclidean distance between the ending frame of a_{t-1} and the starting frame of η. If $L(p_t) \wedge T(a_{t-1})$ isn't empty, this method can ensure the co-articulation by finding the AnimPho a_t that's naturally connected to a_{t-1} in the captured dataset. However, it doesn't work well in practical applications, primarily because the missing rate of this method is typically very high because of the limited size of the captured dataset. As a result, the synthesized animations are often incorrectly articulated and oversmoothed (that is, only minimizing the smoothness energy defined in Equation 1).

Figure 4 illustrates a simple example of this naive AnimPho selection strategy. Let's assume we initially select the third AnimPho for phoneme p_1. When p_2 comes, we find that none of the AnimPhos in $L(p_2)$ can connect to the third AnimPho in the dataset, so we select the one with a minimum smoothness energy (assuming it's the sixth AnimPho). When p_3 comes, again we find that none of the AnimPhos in $L(p_3)$ can connect to the sixth AnimPho, so we select an AnimPho (assuming it's the seventh one) based on the smoothness energy. The missing rate of this simple example would be

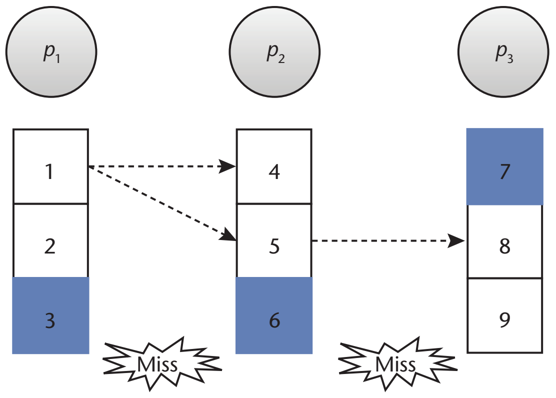

Figure 4. Naive AnimPho selection strategy. The circles on the top are the phonemes that come to the system sequentially. The square below each phoneme p_i is $L(p_i)$, and the number in the squares denotes the index of AnimPho. Dotted directional lines denote existing AnimPho transitions in the dataset. The green AnimPhos are selected based on the naive AnimPho selection strategy.

100 percent, and the natural AnimPho connection information (dotted directional lines in Figure 4) wouldn't be exploited.

To solve both the oversmoothness and the inaccurate articulation issues, we need to properly utilize the AnimPho connection information, especially when the missing rate is high. In our approach, we compute the priority value, v_i, for each AnimPho $a_i \in L(p_t)$ based on the precomputed transition information. Specifically, the priority value of an AnimPho a_i is defined as the length of the longest connected AnimPho sequence corresponding to the observed phoneme sequence (up to p_t). The priority value v_i of a_i is calculated as follows:

$$v_i \begin{cases} \max(v_j + 1) & \exists a_j \in L(p_{t-1}), a_i \in T(a_j) \\ 1 & \text{Otherwise} \end{cases}. \tag{2}$$

Then, we select AnimPho a_t by taking both the priority value and the smoothness term into consideration. In our approach, we prefer to select the AnimPho with the highest priority value; if multiple AnimPhos have the same high-priority values, we select the smoothest one when it's connected with a_{t-1}.

We use the same illustrative example in Figure 5 to explain our algorithm. We start by selecting the third AnimPho for p_1. When p_2 comes, we first compute the priority value for each AnimPho in $L(p_2)$ by using the AnimPho transition information. The priority value of the fourth AnimPho, v_4, is 2, because the AnimPho sequence <1, 4> corresponds to the observed phoneme sequence <p_1, p_2>. Similarly, we can compute the priority

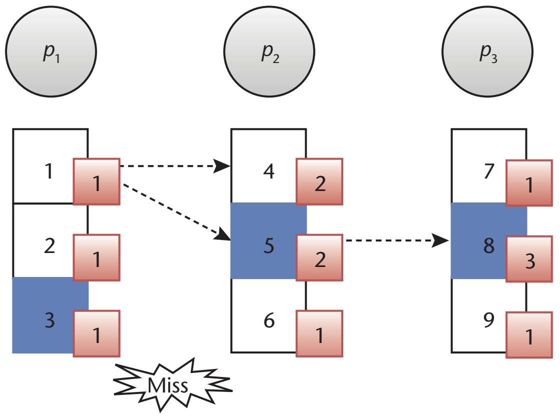

Figure 5. Priority-incorporated AnimPho selection strategy. Numbers in red boxes are the AnimPho priority values.

value of the fifth AnimPho as 2. After we compute priority values, we select one AnimPho for p_2. Although we find that none of the AnimPhos in $L(p_2)$ is connected to the third AnimPho, instead of selecting the sixth AnimPho that minimizes the smoothness term, we choose the fourth or the fifth AnimPho (both have the same higher priority value). Assuming the fifth AnimPho has a smaller smoothness energy than the fourth AnimPho, we select the fifth for p_2. When phoneme p_3 comes, the eighth AnimPho has the highest priority value, so we simply select it for p_3.

As Figure 6 shows, compared with the naive AnimPho selection strategy, our approach can significantly reduce the missing rate. Still, the missing rate in our approach is substantial (around 50 percent), for two reasons:

- *The unpredictability of the next phoneme*. At the AnimPho selection step in live speech-driven lip-sync, the next (forthcoming) phoneme information is unavailable, so we can't predict which AnimPho would provide the optimal transition to the next phoneme.
- *The limited size of the motion dataset*. In most practical applications, the acquired facial motion dataset typically has a limited size due to data acquisition costs and other reasons. For example, the average number of outgoing transitions from an AnimPho in our dataset is 3.45. Therefore, once we pick an AnimPho for p_{t-1}, the forward AnimPho transitions for p_t are limited (that is, on average 3.45).

In theory, to dramatically reduce the missing rate (get close to zero), these two issues must be well taken care of. In this work, rather than capturing a much larger facial motion dataset to reduce the missing rate, we focus on how to utilize a limited motion dataset to produce acceptable live speech-driven lip-sync results. The motion-blending step can handle the discontinuity incurred by the missing event.

Motion Blending

In this section, we describe how to generate the final facial animation by using a motion-blending technique based on the selected AnimPho a_t and the duration of phoneme p_t (which is output from a live speech phoneme-recognition system such as Julius). Specifically, this technique consists of three steps: selecting an AnimPho b_t that both connects to a_{t-1} and has the most similar motion trajectory with a_t; generating an intermediate motion segment m_t based on a_{t-1}, a_t, and b_t; and doing motion interpolation on m_t to produce the desired number of animation frames for p_t, based on the inputted duration of p_t.

Selection of b_t. As mentioned earlier, at runtime, our live speech-driven lip-sync approach still has a nontrivial missing rate (on average, about 50 percent) at the AnimPho selection step. To solve the possible

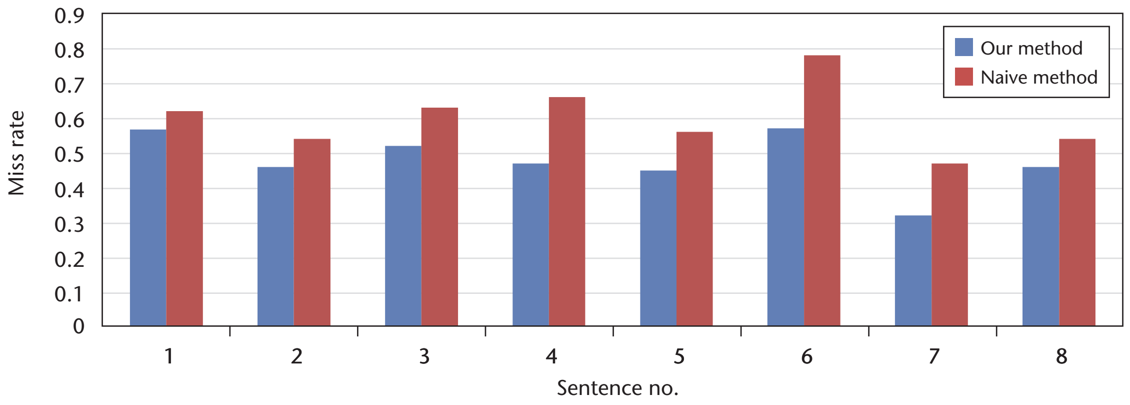

Figure 6. The naive AnimPho selection strategy versus our approach. The naive approach has a higher missing rate when tested on the same eight phrases.

motion discontinuity issue between a_{t-1} and a_t, we need to identify and utilize an existing AnimPho transition <a_{t-1}, b_t> in the dataset such that b_t has the most similar motion trajectory with a_t. Our rationale is that the starting frame of b_t can ensure a smooth connection with the ending frame of a_{t-1}, and b_t can have a similar motion trajectory with a_t in most places. We identify the AnimPho b_t by minimizing the following cost function:

$$b_t = \underset{\eta \in T(a_{t-1})}{\arg\min}\, \text{dist}\left(\eta, a_t\right), \tag{3}$$

where dist(η, a_t) returns the Euclidean distance between a_t and η. Note that if $a_t \in T(a_{t-1})$, which means a natural AnimPho transition exists from a_{t-1} to a_t in the dataset, Equation 3 is guaranteed to return $b_t = a_t$.

Generation of an intermediate motion segment, m_t. After b_t is identified, we need to modify it to ensure the motion continuity between the previous animation segment a_{t-1} and the animation for p_t that we're going to synthesize. Here, we use b_t' to denote the modified version of b_t. We generate an intermediate motion segment m_t by linearly blending b_t' and a_t.

All the AnimPhos (including b_t, a_{t-1}, and a_t) store the center of the corresponding motion segment clusters, as described earlier; they don't retain the original motion trajectory in the dataset. As a result, the starting frame of b_t and the ending frame of a_{t-1} may be slightly mismatched, although the AnimPho transition <a_{t-1}, b_t> indeed exists in the captured motion dataset (Figure 7a). To make the starting frame of b_t perfectly match with the ending frame of a_{t-1}, we need to adjust (via a translation transformation) b_t by setting the starting frame of b_t equivalent to the ending frame of a_{t-1} (Figure 7b).

Then, we linearly blend two motion segments, b_t' and a_t, to obtain an intermediate motion segment m_t, as Figure 7c illustrates. As described earlier, we use give evenly sampled frames to represent a motion segment. Therefore, this linear blending can be described as follows.

$$m_t[i] = \left(1 - \frac{i-1}{4}\right) \times b_t'[i] + \frac{i-1}{4} \times a_t[i], (1 \le i \le 5), \tag{4}$$

where $m_t[i]$ is the ith frame of motion segment m_t.

Motion interpolation. By taking the downsampled motion segment m_t as input, we use an interpolation method to compute the final animation frames

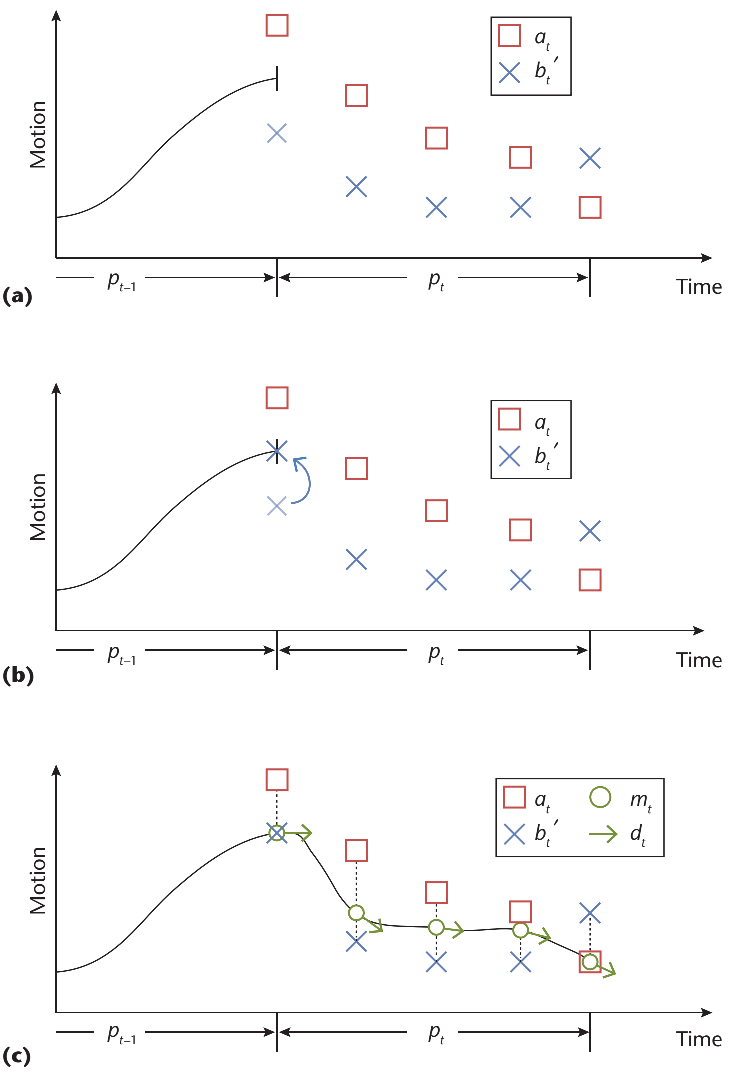

Figure 7. The motion-blending and interpolation process. (a) After the AnimPho selection step, we can see that either a_t or b_t has a smooth transition from the previous animation segment (denoted as the black solid line). (b) We set the starting frame of b_t equivalent to the ending frame of the previous animation segment (for phoneme p_{t-1}). (c) We linearly blend a_t and b_t' to obtain a new motion segment m_t and further compute the derivatives of m_t. Finally, we apply the Hermit interpolation to obtain in-between frames.

for phoneme p_t based on its duration. We calculate the required number of frames N in final animation as N = duration(p_t) × FPS, where duration(p_t) denotes the duration of phoneme p_t and FPS is the resulting animation's frames per second.

In this work, we use the Hermit interpolation to compute in-between frames in two sample frames. The derivatives of the five sample frames are computed as follows:

$$d_t[i] = \begin{cases} d_{t-1}[5] & \text{if } i = 1 \\ \left(m_t[i+1] - m_t[i-1]\right)/2 & \text{if } 1 < i < 5 \\ m_t[i] - m_t[i-1] & \text{if } i = 5 \end{cases}, \tag{5}$$

```
Input: Sample frames, $m_t$; the derivatives of sample frames, $d_t$; the target
        number of frames, $N$.
Output: The interpolated sequence, $S_t$.
  1: for i = 1 $\rightarrow N$ do
  2:     Find the upper and lower sample frame [lower, upper]= Bound($i$)
  3:     $\alpha$ = ($i$ - lower)/(upper - lower)
  4:     $S_t[i]$ = HermitInterpolation(upper, lower, $\alpha$)
  5: end for
```

Figure 8. Algorithm 1. For motion interpolation, we take the upper and lower sample frames as input and then compute the in-between frame corresponding to $\alpha \in$ [0, 1].

Table 1. Three cases/configurations for our lip-sync experiments.

Aproach	Live speech phoneme recognition	Real-time facial motion synthesis
Case 1	Not used	Yes
Case 2	Yes	Yes
Baseline	Not used	Yes

where d_{t-1} is the derivatives of the previous downsampled motion segment. As illustrated in Figure 7c, the Hermit interpolation can guarantee the resulting animation's C^2 smoothness. Finally, we use m_t and d_t to interpolate in-between animation frames. The motion interpolation algorithm can be described using Algorithm 1 (see Figure 8). Specifically, function `p=HermitInterpolation(upper, lower,` α`)` takes the upper and lower sample frames as input and computes the in-between frame corresponding to $\alpha \in$ [0, 1].

Results and Evaluations

We implemented our approach using C++ without GPU acceleration (see Figure 9). The off-the-shelf test computer we used in all our experiments is an Intel core I7 2.80-GHz CPU with 3 Gbytes of memory. The real-time phoneme recognition system is Julius, which we trained with the open source VoxForge acoustic model. In our system, the 3D face model consists of 30,000 triangles, and a thin-shell deformation algorithm is used to deform the face model based on synthesized facial marker displacements. To improve the realism of synthesized facial animation, we also synthesized corresponding head and eye movements simultaneously by implementing a live speech-driven head-and-eye motion generator.[6]

Quantitative Evaluation

To quantitatively evaluate the quality of our approach, we randomly selected three speech clips with associated motion from the captured dataset and computed the root mean squared error (RMSE) between the synthesized motion and the captured motion (only taking markers in the lower face into account, as illustrated in Figure 2).

To better understand the accuracy of our approach in different situations, we synthesized motion in three cases, as shown in Table 1. Here, live speech phoneme recognition means that the phoneme sequence is sequentially outputted from Julius by simulating the prerecorded speech as live speech input. Real-time facial motion synthesis means the facial motion is synthesized phoneme by phoneme, as described earlier. Our approach was used as the synthesis algorithm in cases 1 and 2.

We also compared our approach with a baseline method[1] that adopts the classical Cohen-Massaro co-articulation model[7] to generate speech animation frames via interpolation. This method also needs users to manually specify (or design) visemes, which are a set of static mouth shapes that represent different phonemes. To ensure a fair comparison between our approach and the chosen baseline, the visemes used in the baseline's implementation[1] were automatically extracted as the center (mean) of the cluster that encloses all the representative values of the corresponding phonemes.

Table 2 shows the comparison results. Case 2 has the largest RMSE errors in the three test cases due to the inaccuracy of live speech phoneme recognition. The RMSE error of case 1 is smaller than that of the baseline approach in all three test cases, but the animation results in case 1 are clearly better than those in the baseline approach in terms of motion smoothness and visual articulation.

Runtime Performance Analysis

To analyze our approach's runtime efficiency, we used three prerecorded speech clips, each of which is about two minutes in duration. We sequentially fed the speech clips into our system by simulating them as live speech input. During the live speech-driven lip-sync process, our approach's runtime performance was recorded simultaneously. Table 3 shows the breakdown of its per-phoneme computing time. From the table, we can make two observations.

First, the real-time phoneme recognition step consumes about 60 percent of our approach's total computing time on average. Its average motion synthesis time is about 40 percent (around 40.85 ms) of the total computing time, which indicates that our lip-sync algorithm itself is highly efficient.

Second, when we calculated the average phoneme number per second in the test data and the average FPS, we computed the average FPS as follows:

$$\text{Average FPS} = \frac{\sum_{i=1}^{N} \frac{\text{Number of Frames}(p_i)}{\text{Total Compute Time}(p_i)}}{N}, \quad (6)$$

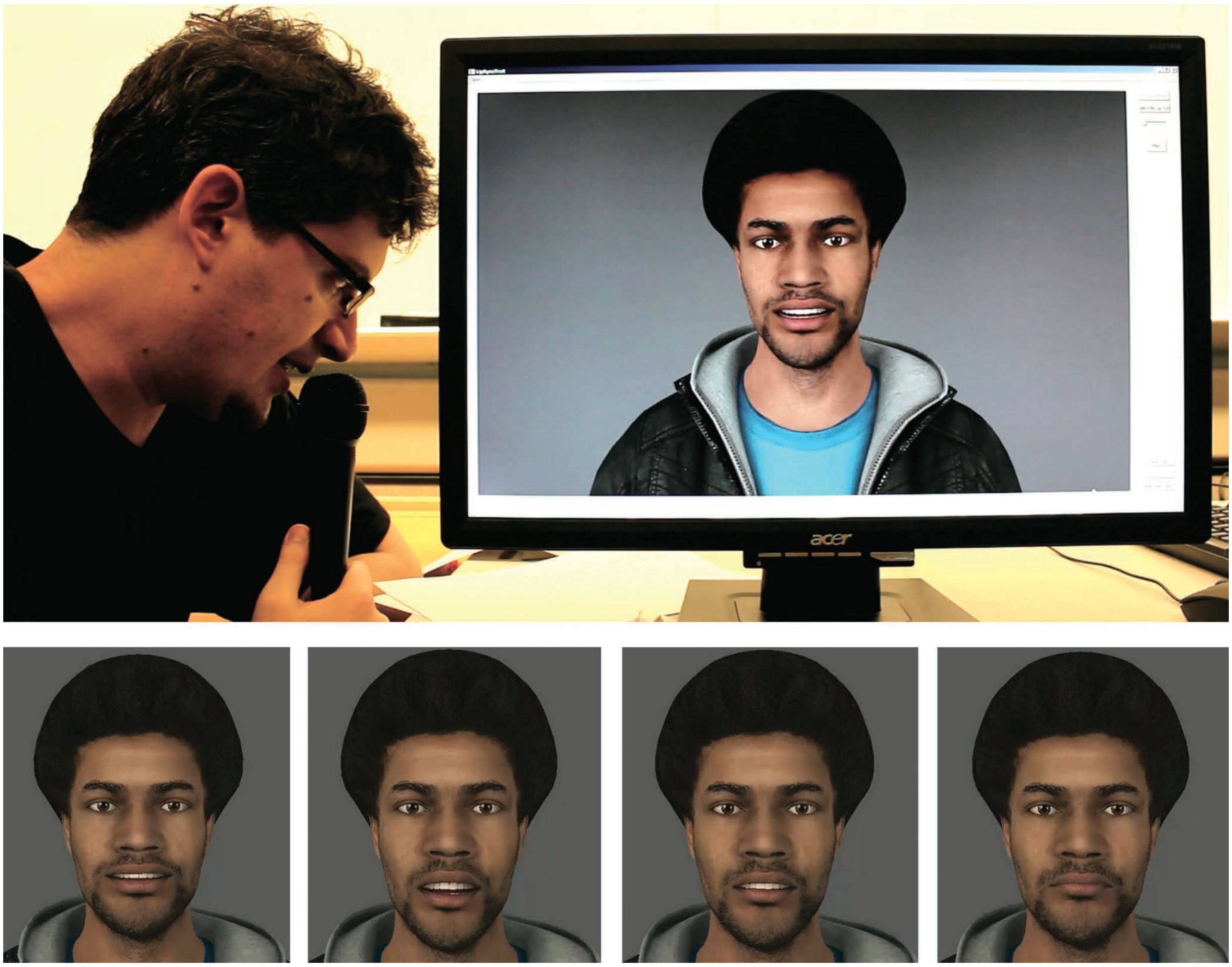
Figure 9. Experimental scenario. Several selected frames synthesized by our approach.

where N is the total number of phonemes in the test data; Number of Frames(p_i) denotes the number of animation frames needed to visually represent p_i, which can be simply calculated as its duration (in seconds) multiplied by 30 frames (per second) in our experiment; and Total Computing Time(p_i) denotes the total computing time used by our approach for phoneme p_i including the real-time phoneme recognition time and motion synthesis time. As Table 3 shows, the calculated average FPS of our approach is around 27, which indicates that our approach is able to approximately achieve real-time speed on an off-the-shelf computer, without GPU acceleration.

Table 2. RMSE of the three test speech clips.

Aproach	Audio 1	Audio 2	Audio 3
Case 1	18.81	19.80	25.77
Case 2	24.16	25.79	27.81
Baseline	22.34	22.40	26.17

On the one hand, our approach is simple and efficient and can be straightforwardly implemented. On the other, it works surprisingly well for most practical applications, achieving a good balance between animation realism and runtime efficiency, as demonstrated by our results. Because our approach is phoneme-based, it can naturally

Table 3. Breakdown of our approach's per-phoneme computing time.

Test audio clip	Phoneme recognition time (ms)	Motion synthesis (ms)	Motion blending (ms)	Average total time (ms)	Average phonemes per second	Average frames per second (FPS)
1	76.73	38.56	0.16	128.28	8.84	25.62
2	90.13	39.10	0.14	144.40	7.29	30.03
3	93.80	44.89	0.15	142.23	7.43	26.77

handle speech input from different speakers, assuming the employed state-of-the-art speech recognition engine can reasonably handle the speaker-independence issue.

Despite its effectiveness and efficiency, however, our current approach has two limitations. One, the quality of visual speech animation synthesized by our approach substantially depends on the efficiency and accuracy of the used real-time speech (or phoneme) recognition engine. If the speech recognition engine can't recognize phonemes in real time or does it with a low accuracy, then such a delay or inaccuracy will be unavoidably propagated to or even be exaggerated in the employed AnimPho selection strategy. Fortunately, with the continuous improvement of state-of-the-art real-time speech recognition techniques, we anticipate that our approach will eventually generate more realistic live speech-driven lip-sync without modifications. Two, our current approach doesn't consider affective state enclosed in live speech and thus can't automatically synthesize corresponding facial expressions. One plausible solution to this problem would be to utilize state-of-the-art techniques to recognize the change of a subject's affective state in real time based on his or her live speech alone; such information could then be continuously fed into an advanced version of our current approach that can dynamically incorporate the emotion factor into the expressive facial motion synthesis process.

As future work, in addition to extending the current approach to various mobile platforms, we also plan to conduct comprehensive user studies to evaluate its effectiveness and usability. We also want to investigate better methods to increase phoneme recognition accuracy from live speech and to fiducially retarget 3D facial motion capture data to any static 3D face models.

Acknowledgments

This work was supported in part by NSF IIS-0914965, NIH 1R21HD075048-01A1, and NSFC Overseas and Hong Kong/Macau Young Scholars Collaborative Research Award (project 61328204). We thank Bingfeng Li and Lei Xie for helping with the Julius system and for numerous insightful technical discussions. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the agencies.

References

1. B. Li et al., "Real-Time Speech Driven Talking Avatar," *J. Tinghua Univ. Science and Technology*, vol. 51, no. 9, 2011, pp. 1180–1186.
2. Y. Xu et al., "A Practical and Configurable Lip Sync Method for Games," *Proc. Motion in Games* (MIG), 2013. pp. 109–118.
3. P. Hong, Z. Wen, and T.S. Huang, "Real-Time Speech-Driven Face Animation with Expressions Using Neural Networks," *IEEE Trans. Neural Networks*, vol. 13, no. 4, 2002, pp. 916–927.
4. R. Gutierrez-Osuna et al., "Speech-Driven Facial Animation with Realistic Dynamics," *IEEE Trans. Multimedia*, vol. 7, no. 1, 2005, pp. 33–42, 2005.
5. Z. Deng and U. Neumann, "eFASE: Expressive Facial Animation Synthesis and Editing with Phoneme-Level Controls," *Proc. ACM SIGGGRAPH/Eurographics Symp. Computer Animation*, 2006, pp. 251–259.
6. B. Le, X. Ma, and Z. Deng, "Live Speech Driven Head-and-Eye Motion Generators," *IEEE Trans. Visualization and Computer Graphics*, vol. 18, no. 11, 2012, pp. 1902–1914.
7. M.M. Cohen and D.W. Massaro, "Modeling Coarticulation in Synthetic Visual Speech," *Models and Techniques in Computer Animation*, Springer, 1993, pp. 139–156.

Li Wei *is a PhD student in the Department of Computer Science at the University of Houston. His research interests include computer graphics and animation. Wei received a BS in automation and an MS in automation from Xiamen University, China. Contact him at xmuweili@gmail.com.*

Zhigang Deng *is an associate professor of computer science at the University of Houston. His research interests include computer graphics, computer animation, and human-computer interaction. Deng received a PhD in computer science from the University of Southern California. Contact him at zdeng4@uh.edu.*