Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2004)
R. Boulic, D. K. Pai (Editors)

Real-time Speech Motion Synthesis from Recorded Motions

Yong Cao[1,2] Petros Faloutsos[1] Eddie Kohler[1] Frédéric Pighin[2]

[1]University of California at Los Angeles, Department of Computer Science
[2]University of Southern California, Institute for Creative Technologies

Abstract

Data-driven approaches have been successfully used for realistic visual speech synthesis. However, little effort has been devoted to real-time lip-synching for interactive applications. In particular, algorithms that are based on a graph of motions are notorious for their exponential complexity. In this paper, we present a greedy graph search algorithm that yields vastly superior performance and allows real-time motion synthesis from a large database of motions. The time complexity of the algorithm is linear with respect to the size of an input utterance. In our experiments, the synthesis time for an input sentence of average length is under a second.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Animation, F.2.2 [Non-numerical Algorithms and Problems]: Pattern matching.

1. Introduction

Realistic facial animation remains a very challenging problem in computer graphics. The human face is the most complex muscular region of the human body. Hundreds of individual muscles contribute to the generation of complex facial expressions and speech. Even though the dynamics of each of these muscles is well understood, their combined effect is very difficult to simulate precisely. Motion capture allows the recording of high fidelity facial motions. But this technique is mostly useful for specific shots since the recorded motions are difficult to modify. Editing motion capture data often involves careful key-framing by a talented animator. Motion capture by itself cannot be used for automated facial animation.

This issue has spurred a great deal of interest for data-driven or machine learning approaches. In these approaches, the motions of the face are no longer viewed as the results of a complex bio-mechanical system but rather as valuations of an abstract function. In such framework, this function can be approximated using a training set of sample values. The strength of data-driven approaches is to provide a yardstick against which to compare the synthesized motions: the quality of synthesized motions can be evaluated by how much they deviate from the data. 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.

Data-driven approaches have yielded some of the most high fidelity facial animation systems to date. However, most of this work has focused on the issue of realism. Little has been done regarding real-time facial animation. In particular, many data-driven algorithms are based on a database search: the input audio is segmented into a sequence of speech labels (e.g., phonemes or visemes) that are used to find corresponding motion segments. Some of these algorithms use graph structured databases and search algorithms whose complexity depends exponentially on the duration of the input speech. These techniques are clearly inappropriate for real-time applications.

This performance issue is a very practical one since real-time facial animation has many applications. For instance, many computer games feature speaking three-dimensional digital humans. For a game, it might be possible to use pre-recorded animation, however it is not an option for digital chat-room avatars or a virtual clerk. In these cases, animations have to be generated on the fly to match a spoken or synthesized utterance. A simple solution is to associate a mouth shape to each phoneme or class of phoneme (e.g., visemes) and to interpolate between these shapes. This ap-

proach however yields lower quality motions than motion capture-based systems. In this paper, we manage to reconcile automatic high-fidelity facial animation with real-time performance.

Our approach is based on a novel data structure and an associated real-time search algorithm. The data structure encapsulates the facial motion database along with speech information into a graph that we call *Anime Graph*. Given an input speech, we search the *Anime Graph* for a sequence of motion segments that matches the input audio. Instead of exhibiting exponential complexity, as most graph-based motion synthesis algorithms do, our greedy search algorithm is a linear-time method that has straightforward real-time implementations. In addition, we prove that the algorithm is optimal under reasonable assumptions. The remainder of the paper is organized as follows. Section 2 reviews the related literature. Section 3 describes briefly the data we used for our experiments. Section 4 provides an overview of the general problem. Section 5 introduces our novel facial motion data structure. Section 6 presents a depth-first search algorithm and our novel real-time approach. Section 7 presents our experiments. Section 8 discusses the limitations of our approach and future work. Finally, Section 9 concludes the paper.

2. Previous Work

Facial motions can typically be split into three components: the *lower face motion* (lip and chin), the *upper face motion* (eyes and eyebrows), and the *rigid head motion*. In this work, we focus on *lip-synching*: the synthesis of lip motion matching an input audio sentence. Hence, we focus mostly on the motion of the lower face. What makes this problem difficult is the *co-articulation* effect: the shape of the mouth corresponding to a phoneme depends on the phonemes that come before and after the given phoneme. Studies have shown that co-articulation may affect mouth shape of 2-5 neighbor phonemes. Considering that the English language has typically 46 distinct phonemes, it is not practical to solve the co-articulation problem using simple lookup tables.

A simple solution to this problem is to model co-articulation as a set of rules [Pel91, CPB*94]. However, a complete set of such rules does not exist. [KMG02] base their co-articulation model on a limited set of phonemes that appear to be visually more important than others, such as vocals and labial consonants. The mouth shape of these phonemes is kept fixed or has little variations.

Generally, approaches that attempt to solve lip-synching problem fall in three categories.

The physics-based approach uses the laws of physics and muscle forces to drive the motion of the face. Although it is computationally expensive, it has been shown to be quite effective [LTW95, Wat87].

Data-driven approaches use an input speech signal to search a database of speech-indexed motions for the closest match. *Video Rewrite* [BCS97] is a representative example of such techniques. It relies on a database of motions segmented into triphones. A new audiovisual sequence is constructed by concatenating the appropriate triphones from the database. This method requires a large database which leads to a scaling problem. In addition, the use of triphones only allows a limited co-articulation model. Instead of relying on phonemic information, [CXH03] uses vision-based control to drive 3D faces. At first, the trajectories of a limited set of control parameters are extracted from video using vision-based tracking. These trajectories are then translated into high quality motions by searching a database of pre-recorded motion capture data. However, the speech motion database is limited and the system does not take speech as an input. As the result, co-articulation is not well preserved.

A third class of techniques attempts to eliminate the need for large example databases by creating compact statistical models of face motion. Hidden-Markov and Gaussian mixture models are two machine learning techniques that are frequently used for this problem [BS94, MKT*98, CM93]. For instance, *Voice Puppetry* [Bra99] develops a mapping from voice to face by learning a model of a face's observed dynamics. The model takes into account the position and the velocity of facial features and learns a probability distribution over the different facial configurations. The training data is 180 seconds of video at a sampling rate of 29.97Hz which has problems with plosives and short duration phonemes. [EGP02] develops a variant of the *Multidimensional Morphable Model* (MMM), which is represented as a set of optical flow vectors. It is used to describe images with local variations in shape and appearance. This model can be applied to statistically interpolate novel video frames corresponding to input speech segments. First, they construct a sparse adjacency matrix of the video frames and compute shortest paths between pairs of frames. Then the entire corpus is projected on an MMM model and the shortest paths between images become trajectories in the MMM-space. Synthesizing the MMM-trajectories for a new input sentence is formulated as a regularization problem and takes on average of 7 seconds on a Pentium 450MHz machine. Although the technique could extend to 3D models, it has so far been tested only on the 2D cases. [SBCS04] learn a linear dynamical system from recorded speech video clips. The system is driven by both deterministic speech input and an unknown stochastic input. Because of the limitation of the model, only video clips for single word or very short sentences can be synthesized. Therefore, co-articulation can not be fully modeled in this approach.

Our work is inspired by the graph-based approaches to full-body motion synthesis [KGP02], [LWS02], [LCR*02], [AF02]. [KGP02] uses a branch-and-bound algorithm to search a connected graph for a suitable path that satisfies the constraints. To deal with the exponential complexity of the algorithm, they employ a more efficient incremental search

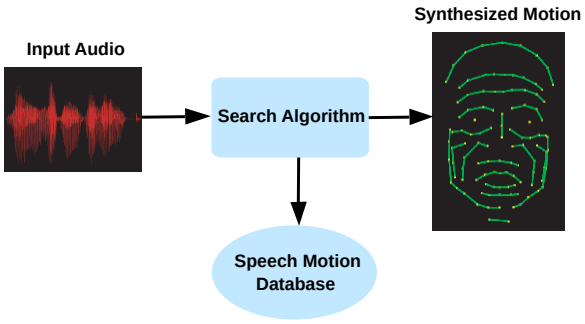

Figure 1: *High level overview of our approach.*

approach. [AF02] uses a hierarchy of graphs and a randomized search algorithm to sequence motion clips interactively but not in real-time. [LCR*02] use clustering and first-order Markov process to represent their data. The user can search for the most probable motion using three different interfaces. The approach is interactive but not real-time. [LWS02] proposed a two-level statistical model to represent compactly recorded motions.

In this paper, we present a motion graph-based lip-synching algorithm that solves the co-articulation problem in real-time. Unlike most previous methods, our approach synthesizes facial motion in real-time and works with three dimensional motion data.

3. Data Collection and Preprocessing

We have recorded a set of facial motions using a Vicon8 optical motion capture system. We used 109 markers to sample the motion of the face fairly densely. The sampling rate of the data is 120 frame/sec. To drive a 3D textured face mesh, the markers are mapped to corresponding mesh points, and the rest of the mesh is deformed using *Radial Basis Functions* [Buh03].

The entire database sums up to 53 minutes of recorded motions. The shortest sentence is a few seconds long, while the longest sentence has a duration of 15 seconds. Cleaning up the data required about two man months of work.

4. Overview

Given a spoken sentence as input, our goal is to produce a matching facial motion. We rely on a database of speech related recorded facial motions. Our algorithm proceeds in three main steps:

1. The input audio is segmented into a string of speech tokens, called phonemes.
2. The motion database is searched for a set of continuous motion segments matching the phonemes.
3. The motions segments are stitched together to produce a seamless motion.

Figure 1 shows an overview of our system. In the following sections, we describe the organization of the motion database and our search algorithm in detail.

5. Motion Database

In this section, we explain how we organize the data into a novel data structure suitable for search-based lip-synching.

5.1. Data segmentation

Our dataset consists of audio and motion data for a large set of sentence-long utterances that vary in length, emotion, and content. We first segment each sentence into phonemes using the *Festival* [SG] software. Since the audio is synchronized with the motion data, we can easily extract the motion segments that correspond to each phoneme segment in a sentence. To reduce the size of our database, we compress the motion curves using *Principal Components Analysis*. In our experiments, we keep 5 principal components (these cover more than 95% of the variance of the original motion). For each phoneme in the database, we also compute an audio feature vector that we use during the search phase. These audio feature curves consist of the first 9 parameters returned by a RASTA-PLP filter [Int].

To organize the database, each recorded sentence is converted into a sequence of nodes, which we call *animes*. An *anime*, $A =< P, \mathbf{C}, \mathbf{M} >$, captures a phoneme instance and contains a phoneme label P, the associated motion fragment, $\mathbf{M}$, and audio feature vector, $\mathbf{C}$. Like a *viseme*, an anime is the visual counterpart of a phoneme. Unlike a viseme, that is associated with a static mouth shape, an anime is associated with a fragment of face motion.

5.2. The Anime Graph

The shape of the lower face during speech at a specific point in time does not only depend on the current phoneme but also on past and future phonemes (co-articulation). In our framework, we translate this constraint by organizing the set of animes into two main data structures that model contextual information, the ***Anime Graph*** and the ***Anime Array***. The Anime graph keeps the recording order of each anime sequence in the training dataset. The Anime array keeps for each phoneme label a list of all associated animes.

We construct the Anime Graph as follows. If two animes A_i and A_j appear sequentially, we create an edge $A_i \rightarrow A_j$ that reflects the recording order. Thus, each anime sequence in the database is converted to a directed link list. The Anime graph is the collection of all these link lists, as shown in Figure 2.

If n_a is the total number of animes, we can formally define the Anime Graph as follows:

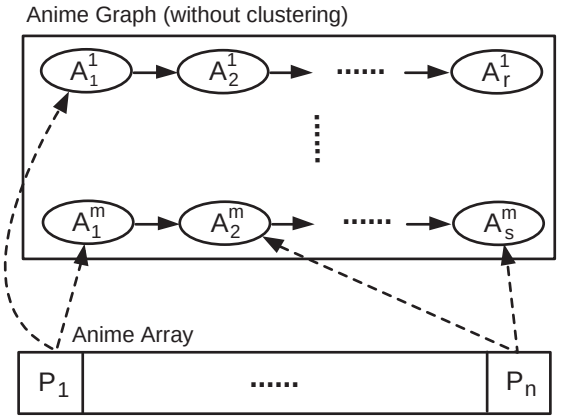

Figure 2: *Anime Graph and Anime Array.*

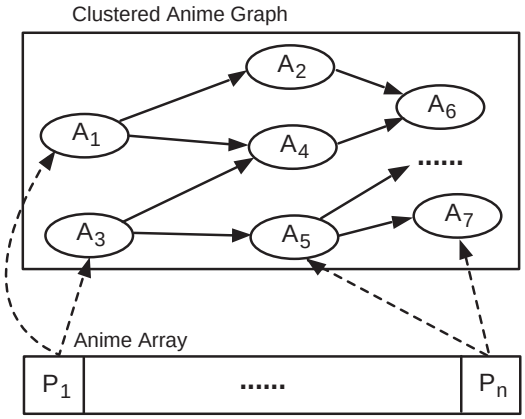

Figure 3: *Anime Graph and Anime Array after clustering and merging.*

$$\begin{aligned} AG &= < Animes, Edges >, \\ Animes &= \{A_i\}, (1 \le i \le n_a), \\ Edges &= \{E_i : A_s \rightarrow A_t\}, (1 \le s,t \le n_a). \end{aligned} \quad (1)$$

5.3. Clustered Anime Graph

The number of animes in the database directly affects the efficiency of the lip-synching process. In our experiments, we use a database of 246 sentences and 7256 animes. To shrink our database and improve performance, we reduce the number of animes through clustering.

Clustering. A careful examination of the motion fragment of animes in our database shows that many are similar despite having different audio curves or phoneme labels. Intuitively speaking, the same lip motion often corresponds to different phonemes since speech is not formed by lip-motion alone. The same phenomenon allows the association of multiple phonemes to the same viseme.

To take advantage of these similarities, we first normalize the duration of the motion fragment of each anime to compare them more easily. We then consider each of the normalized fragment as a vector V_i in a high dimensional space (465 dimensions in our experiments). We then find clusters within the set $\{V_i : i = 1,\ldots,n\}$, where n is the total number of animes, using a *K-Means* clustering algorithm. Choosing the number of clusters allows us to trade off quality for efficiency. Our experiments show that using 1000 clusters we achieve a balance between quality and efficiency.

Merging. For each cluster, we select a representative motion fragment that is closest to the cluster's mean. We then replace the animes in this cluster with a single anime. This new anime contains a single motion fragment, a list of phoneme labels and a set of audio features. For example, if a cluster contains only three animes $A_1 =< P_1, \mathbf{C}_1, \mathbf{M}_1 >$, $A_2 =< P_2, \mathbf{C}_2, \mathbf{M}_2 >$, and $A_3 =< P_3, \mathbf{C}_3, \mathbf{M}_3 >$ after merging the corresponding anime is: $A =< \{P_1, P_2, P_3\}, \{\mathbf{C}_1, \mathbf{C}_2, \mathbf{C}_3\}, \mathbf{M}_1 >$ assuming that $\mathbf{M}_1$ is the closest to the mean. In the rest of this paper, we use the following notation for the components of an anime, A:

$$A =< P, \mathbf{C}, \mathbf{M} >, \quad (2)$$

where P is a set of phoneme labels, $\mathbf{C}$ is a set of audio features, and $\mathbf{M}$ is a motion fragment.

After merging, each representative anime of a cluster retains the connections of the original animes. Thus, the resulting Anime Graph becomes a directed connected graph as shown in Figure 3.

In the following, the term Anime Graph refers to either version of the Anime Graph (with clustering or without clustering). When necessary, we explicitly state which version we refer to.

6. Search algorithms for lip-motion synthesis

Given a novel input utterance, we synthesize a matching facial motion by searching for an appropriate path in the Anime Graph.

We first segment the input audio into phonemes using Festival [SG]. For each phoneme, we extract a set of audio features, $\mathbf{C}$, just like we did for the animes. We thus transform the input sentence into a *search sequence*,

$$S_Q = Q_1 \ldots Q_l =< P_1, \mathbf{C}_1 > \ldots < P_l, \mathbf{C}_l > \quad (3)$$

where each node Q holds a phoneme label P_i and associated audio feature curves, $\mathbf{C}_i$. The goal of the search algorithm is to find a corresponding anime sequence $S_A = A_1 A_2 \ldots A_l$ that best matches S_Q.

Each anime node in S_A should have the correct phoneme

label and good co-articulation. Evaluating how well a synthesized motion models co-articulation is difficult. By construction, anime nodes are connected within the Anime Graph if and only if they correspond to joint motion segments within a recorded utterance. Thus, connected animes are guaranteed to have correct co-articulation. Therefore to ensure the best possible co-articulation, we need to find a sequence of animes with the minimum number of disconnected nodes (jumps). When more than one matching sequence has the same number of jumps, we can use the audio features to break the tie.

In summary, the search algorithm should use the following three criteria:

1. Phoneme matching: $Q_i.P \in A_i.P$.
2. Minimum number of *Jumps*.
3. Audio feature curves matching.

The first criterion requires that the phoneme labels are the same. The second criterion enforces co-articulation: by using continuous segments of motions that are as long as possible, we maximize the amount of contextual information mined from the database. The last criterion helps enforce additional speech constraints.

The next section presents a depth-first algorithm that given a search sequence, S_Q (Equation 3), uses the above criteria to find the best matching anime sequence, S_A. Section 6.2 proposes a novel greedy search algorithm that outperforms current depth-first search approaches by orders of magnitude and achieves real-time performance.

6.1. Depth-first graph search algorithm

Given a search sequence S_Q, we use a depth-first search approach and the above criteria to search the Anime Graph for a matching anime sequence S_A. In particular, we use a *Priority-Queue* implementation of the *Branch and Bound* search algorithm. This algorithm, called *PQ-DFS*, finds the global optimal anime sequence with respect to the 3 criteria we described above. However, the size of the Anime Graph and the length of the search sequence prevent it from running in real time. To improve the efficiency of the search, we split the search sequence, S_Q, into a set of sub-sequences of maximum length h. We then perform a search for each subsequence separately and at the end concatenate the resulting anime sequences. Thus, we trade off quality for efficiency and find a local optimum that consists of shorter optimal sequences. In our experiments, we choose h between 3 and 6. Here is the *PQ-Matching(h)* algorithm in detail:

Algorithm 1 *PQ-Matching(h,S_Q).*

Input: search bound h, search sequence $S_Q = Q_1Q_2...Q_l$
Output: anime sequence $S = A_1A_2...A_l$

```
i ← 1, S_A ← ∅
while i ≤ l do
    A_i ... A_{i+h-1} ← PQ-DFS (Q_i Q_{i+1} ... Q_{i+h-1})
    S_A ← concat(S_A, A_i)
    i ← i + h_q
end while
return S_A
```

The time complexity of this algorithm is defined by the number of executions of the while loop and the complexity of the step that performs a depth-first-search operation to find the optimal sub-sequence of length h. The average case time complexity of the *PQ-DFS* step is $O((\frac{n}{p})^h)$, where n is the number of anime nodes in the database, and p is the number of phonemes in English (in our experiment $p = 46$). Note that for the Anime Graph after clustering and merging n is the number of clusters. We can chose to advance i by any number h_q between one and h trading off quality for efficiency. In any case, the while loop executes l/h_q times, where l is the length of the input sequence. Thus the average case time complexity of this algorithm is $O((\frac{n}{p})^h \times l/h_q)$. Even with some heuristic speeding-up methods, the time complexity of these Depth-First graph search algorithms are still exponential with respect to the depth (bound) h of the search.

6.2. Greedy search algorithm

We now show that by ignoring the third criterion, we can develop a greedy search algorithm that can find a matching anime sequence, S_A, with the minimum number of jumps. The proposed algorithm is linear with respect to the length of the search sequence and runs in real-time.

By ignoring the audio features, our search problem becomes analogous to a string matching problem. In what follows, we will use the string matching analogy because it simplifies terminology and notation.

Definition 1: (Tile Matching). Given an input string s and a set of strings $\Sigma = \{s_1, s_2, ..., s_n\}$, a tile matching is a set of tiles $T = [\tau_1, \tau_2, ..., \tau_k]$, where:

1. Each tile τ_i is a substring of some string s_j.
2. The concatenation of all the tiles, $\tau_1\tau_2...\tau_k$, equals the input string s.

Definition 2: (Minimum Tile Matching). Given an input string s and a set of strings Σ, a minimum tile matching is a tile matching T that uses as few tiles as possible. That is, every tile matching of s with Σ uses at least as many tiles as T.

To continue our definition, we first introduce a string operator $[\,]$. Given a string s, let $s[i]$ equals the ith character of s, and let $s[i, j]$ equal the substring of s starting at the ith character and continuing through the jth character. So if $s =$ "$abcdef$", then $s[1] =$ "a" and $s[2,4] =$ "bcd".

Definition 3: (Greedy Tile Matching). Given an input

string s and a set of strings Σ, a greedy tile matching is a tile matching T with the following 2 properties. (Assume that $len(\tau_1) = m$, so $\tau_1 = s[1,m]$.)

1. The first tile τ_1 is as long as possible. This means that either $len(s) = m$ (so there are no more characters to match), or $s[1,m+1]$ is not a substring of any string in Σ (so no longer match exists).
2. The remaining tiles form a greedy tile matching of the remaining portion of the string. This means that either $len(s) = m$ (so the whole string marches), or $[\tau_2, ..., \tau_k]$ forms a greedy tile matching of $s[m+1, len(s)]$.

Theorem 1: (Greedy is optimal). Any *Greedy Tile Matching* is also a *Minimum Tile Matching*. (See Appendix A for the proof.)

If we consider the unclustered version of the Anime Graph, it is easy to see that our anime matching problem with minimum number of jumps is analogous to a *Minimum Tile Matching* problem as defined above. Based on Theorem 1, we propose the following greedy search algorithm for Anime matching that finds a matching anime sequence with minimum number of jumps:

Algorithm 2 *GreedyMatching(S_Q).*

Input: search sequence $S_Q = Q_1 Q_2 ... Q_l$
Output: anime sequence $S = A_1 A_2 ... A_l$

```
1:  i ← 1, S_A ← ∅
2:  while i ≤ l do
3:      k ← 0
4:      for each A_j^i : Q_i.P ∈ A_j^i.P do
5:          PH_j ← LongestMatching(A_j^i, Q_i ... Q_l)
6:          k ← k + 1
7:      end for
8:      PH ← longest(PH_1, ..., PH_k)
9:      S_A ← concat(S_A, PH)
10:     i ← i + length(PH)
11: end while
12: return S_A
```

The subroutine $LongestMatching(A^i_j, Q_i \dots Q_l)$ used in the algorithm returns the longest matching path of search sequence $Q_i \dots Q_l$ starting at anime A^i_j. The subroutine $longest(PH_1, \dots, PH_k)$ returns the longest path from a set of k paths $(PH_1, \dots, PH_k)$. If there is a tie, it uses the audio features to resolve it.

The algorithm essentially works as follows. For each search node Q_i, it finds all the k animes, A^i_j, in the Anime Graph that correspond to instances of phoneme $Q_i.P$. These animes are provided by the *Anime Array*, defined in Section 5. For each of the k animes A^i_j, it then finds the longest matching path PH starting from A^i_j. This longest path is appended to the current matching sequence and the algorithm repeats with the search sequence starting at Q_{i+h} where h is the length of PH.

Appendix B shows that the worst case time complexity of the *GreedyMatching* algorithm operating on the Anime Graph before clustering and merging is $O(n \times l)$, where n is the number of anime nodes in database and l is the length of the search sequence. For a given motion capture database, n is constant. Therefore the complexity of the algorithm *GreedyMatching* is linear-time with respect to the length of the input (search) sequence.

It is interesting to note that, after anime clustering and merging, the in-degree and out-degree of the animes in the graph may be greater than 1. In this case, the step at line 5, where we calculate the longest path for each anime A^i_j, becomes a depth-first-search step. In that case, the worst case complexity of subroutine $LongestMatching(A^i_j, Q_i \dots Q_l)$ becomes $O(d^{l-i+1})$ where d is the maximum in-degree or out-degree of the graph. However, in our experiments, d is a small number, typically between 3 to 7. The total search time seems not to be affected by this step as shown in Table 1.

6.3. Post Processing

After searching the motion capture database, producing continuous facial motion requires three more post-processing steps: *time-warping*, *blending* and *smoothing*.

Time warping. The duration of the phonemes of the input search sequence Q_i is in general different from the duration of the output anime sequence A_i after searching. We use dynamic time warping algorithm (DTW) to align the corresponding audio features curves. The resulting warping function is then applied to the associated motion curves.

Blending. The quality of the continuous facial motion depends significantly on how we string together the motion segments especially in the presence of *Jumps*. Connecting the motion segments of two anime nodes A_i and A_{i+1} is trivial if these nodes are connected within the Anime Graph. If not, then they correspond to a *Jump* and may not join smoothly. To deal with discontinuous motion introduced by *Jump*, linear blending is an efficient solution. After linearly blending two motions, M_1 with M_2, which has the same number of frames n, the resulting motion M becomes

$$M[i] = (1 - \frac{i-1}{n-1}) \times M_1[i] + \frac{i-1}{n-1} \times M_2[i],\ (1 \le i \le n),$$

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

Let us look into the case with the presence of *Jumps*. We assume motion $A_i.M$ has p frames and motion $A_{i+1}.M$ has q frames. For such nodes, we search the Anime Graph for other instances of the associated phonemes that might be connected. If such nodes A_m and A_n exist then the associated motion curves $A_m.M$ and $A_n.M$ join properly. They essentially serve as an example of how phoneme $A_i.P$ transitions to $A_{i+1}.P$. We linearly time-warp motions $A_m.M$ and $A_n.M$ to p and q frames respectively. Then we linearly blend motion $A_i.M$ with $A_m.M$ and $A_n.M$ with $A_{i+1}.M$ and concatenate the resulting motions.

When we cannot find a pair of connected anime nodes A_m and A_n, we proceed with the following steps. We collect the next q frames of motion following the animes that proceed A_i in the Anime Graph. We denote these frames as M_i. Similarly, we collect the p frames of motion that precede anime A_{i+1} in the Anime Graph and denote them as M_{i+1}. If such frames do not exist because A_i doesn't have a child anime or A_{i+1} doesn't have a parent anime we create them based on the velocity of the motion curves. We then create the motion curves $A_i.M' =< A_i.M, M_i >$ and $A_{i+1}.M' =< M_{i+1}, A_{i+1}.M >$, where the "$< *, * >$" operator indicates concatenation (sequencing) of motion frames. Motions $A_i.M'$ and $A_{i+1}.M'$ have the same number of frames, $p+q$, and are linearly blended together to produce the final transition from A_i to A_{i+1}.

Smoothing. The blending stage creates continuous motion curves for the entire utterance. However, jump matches often introduce high frequencies that create visible artifacts in the resulting motion. To eliminate them, we apply a low-pass filter. The cut-off frequency of the filter is crucial since it can significantly affect the motion. To ensure that only the undesirable frequencies are eliminated, we learn a suitable cut-off frequency from the data. We scan the entire motion database and for each of the independent components of the motion curves, we identify the range of frequencies that contain 99% of the total energy of that component. The highest frequency of that range is the cut-off frequency of our filter.

7. Experiments and Results

We compare the proposed GreedyMatching algorithm to the Branch-and-Bound algorithm. Specifically, we compare the search time and the quality of the resulting motion. To measure quality, we use both algorithms to synthesize lip motion for utterances for which we have recorded motions. These motions are not part of the training set. Using the RMS distance between the synthesized and the actual recorded motion, we can see which algorithms produces motion that is closer to the observed one.

Tables 1–4 show the performance of the two search algorithms and the quality of the motion produced for the same input sentence. The experiments summarized in Tables 2–4 use the unclustered anime graph.

Performance. Our experiments, summarized in Tables 1–4, show that the *GreedyMatching* algorithm is orders of magnitude faster than the *PQ-matching(h)* algorithm. They also show the exponential complexity of the *PQ-matching(h)* algorithm with respect to *h*. Note that in all our experiments, the *PQ-matching(h)* algorithm cannot reach a minimum set of jumps unless *h* is greater than three.

Table 1 shows that the *GreedyMatching* algorithm running on the unclustered Anime graph (Experiment 1) is 870 times faster than the *PQ-Matching(4)* algorithm even when the latter uses the highly clustered Anime graph (Experiment 4).

Experiment	Number of Clusters	PQ (sec)	Greedy (sec)	Speedup
1	unclustered	53.87	0.015	3591.3
2	1000	14.30	0.016	893.8
3	500	18.20	0.015	1213.3
4	200	13.05	0.016	815.6

Table 1: *Comparison of search time between PQ-Matching(4) and GreedyMatching.*

Search Algorithm	Search time (sec)	Number of Jumps	RMS Distance
GreedyMatching	0.02	5	0.11
PQ-Matching(3)	0.75	7	0.08
PQ-Matching(4)	6.30	5	0.08
PQ-Matching(5)	39.61	5	0.08
PQ-Matching(6)	106.45	5	0.08

Table 2: *The length of the input phoneme sequence is 22.*

Search Algorithm	Search time (sec)	Number of Jumps	RMS Distance
GreedyMatching	0.02	13	0.12
PQ-Matching(3)	2.39	14	0.12
PQ-Matching(4)	18.95	14	0.12
PQ-Matching(5)	649.31	13	0.12

Table 3: *The length of the input phoneme sequence is 38.*

Search Algorithm	Search time (sec)	Number of Jumps	RMS Distance
GreedyMatching	0.02	14	0.11
PQ-Matching(3)	3.30	16	0.09
PQ-Matching(4)	51.19	15	0.11
PQ-Matching(5)	580.20	15	0.10

Table 4: *The length of the input phoneme sequence is 37.*

Quality. The RMS error in Tables 2–4 shows that both algorithms synthesize facial motions that are close to the actual recorded motions. It is interesting to note that the RMS error does not show the distribution of the error over the motion. However, there is no standard visual measure for comparing two motions. Figure 4 shows snapshots of facial motion synthesized using the *GreedyMatching* algorithm. We refer the reader to the accompanying video for a visual verification of our results.

8. Discussion and future work

Our system has several limitations. Like most data-driven approaches, its results depend significantly on the quality of the recorded data and the pre-processing of the data. Although our data is of high quality, it does have certain amount of noise. In addition, the segmentation phase presented in Section 5.1 is crucial. Unfortunately, none of the available phoneme segmentation tools guarantee error-free results. In our experiments, we often come across misaligned phoneme boundaries. However, these problems are not particular to our approach and their solution is not the focus of the presented method.

Our search algorithm considers all jump matches as equivalent and returns the first sequence of phonemes it finds with the minimum number of jump matches. However, certain jump matches may introduce more pronounced visual errors in the facial motion than others. For instance, to pronounce certain plosives, such as "p" and "b", we must start with the mouth closed. In future work, we plan to identify such constraints and apply them to the resulting facial motion.

In this paper, we have not addressed the issue of expressive visual speech. Expression is best understood in terms of a set of emotional states such as *anger, happiness* etc. The emotional state of a speaker is partially encoded in the audio signal. In the future, we plan to investigate ways of modeling the emotional state of the speech and taking it into account during the search phase. Our goal is to produce facial motion that not only exhibits correct co-articulation but also matches the varying emotional state of the input speech signal in real-time.

9. Conclusion

We have presented a real-time, motion capture-based approach for high quality lip-syncing. Our greedy approach is based on a novel data structure, the Anime Graph, and an associated search algorithm. We have also shown that the time complexity of the proposed search algorithm is linear with respect to the number of phonemes in the input utterance. The entire synthesis process takes less than a second for average length sentences.

Our approach is significantly faster compared to standard depth-first-search algorithms. It is suitable for interactive applications that require efficient speech-related facial motion such as video games and virtual reality.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments. This paper was partly funded by the Department of the Army under contract number DAAD 19-99-D-0046. Any opinions, findings and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the Department of the Army.

We would also like to thank Wen C. Tien for his help on this paper. Intel Corp., Microsoft Corp. and ATI Corp. also help us with their generous support through equipment and software grants.

Appendix A: Proof of Theorem 1

Lemma 1: Given a *Greedy Tile Matching* $G = \gamma_1\gamma_2...\gamma_k$ and a *Minimum Tile Matching* $T = \tau_1\tau_2...\tau_l$, for every i, $G[1,i]$ is at least as long as $T[1,i]$, that is $len(G[1,i]) \geq len(T[1,i])$.

PROOF:

We prove it by induction. The base case $i = 0$ is obvious, because $G[1,0]$ and $T[1,0]$ are both empty strings. ($len(G[1,0]) = len(T[1,0]) = 0$).

Inductive step: Assume that $len(G[1,i]) \geq len(T[1,i])$. Then $T[1,i+1]$ cannot be longer than $G[1,i+1]$ ($len(G[1,i+1]) \geq len(T[1,i+1])$); because if it were, then $len(\tau_{i+1}) > len(\gamma_{i+1})$ and τ_{i+1} contains γ_{i+1} as a subset, which means γ_{i+1} was not chosen greedily. (PROVED.)

Theorem 1: (Greedy is optimal) Any *Greedy Tile Sequence Matching* is also a *Minimum Tile Sequence Matching*.

PROOF:

Given a *Greedy Tile Matching* $G = \gamma_1\gamma_2...\gamma_k$ and a *Minimum Tile Matching* $T = \tau_1\tau_2...\tau_l$, Lemma 1 showed that for every i, $len(G[1,i]) \geq len(T[1,i])$. Therefore, for $= l$ $len(G[1,l]) \geq len(T[1,l])$. Since $T[1,l]$ has matched the whole input string s ($len(T[1,l]) = len(s)$), G and T must have the same number of tiles.

Appendix B: Time Complexity of the *GreedyMatching* algorithm before clustering

The algorithm is shown in detail in Section 6.2. Assume that the algorithm takes s iterations of the while loop to find the matching sequence. At each iteration j the algorithm finds the longest connected sub-sequence of length m_j. To do this, at each iteration the algorithm also explores k_j paths (inner for loop). Note that if l is the length of the search sequence then $l = \sum_{j=1}^{s} m_j$.

The worst-case time complexity of the algorithm is as follows. At each iteration, the inner for-loop in the worst case executes $k_j = n$ times where n is the total number of animes in the graph. This actually happens only in the extreme case where anime A^i_j has n instances in the graph. The body of this for loop in the worst case takes m_j steps to find the longest path of length m_j. Therefore for s iterations the inner for-loop costs $\sum_{j=1}^{s}(m_j \times k_j)$.

The only other significant operation is in line 8 of the while-loop that computes the longest path from a set of paths $(PH_1, \ldots, PH_{k_j})$. This operation takes k_j time per iteration j for a total of $\sum_{j=1}^{s} k_j$.

Thus, the total running time T of the *GreedyMatching* algorithm is $\sum_{j=1}^{s}(m_j \times k_j + k_j)$. We can compute the final form of the running time of the algorithm as follows:

$$T = \sum_{j=1}^{s}(m_j \times k_j + k_j) = \sum_{j=1}^{s} m_j \times k_j + \sum_{j=1}^{s} k_j. \quad (4)$$

In the worst case, $k_j = n$ and T becomes

$$T = n \times \sum_{j=1}^{s} m_j + \sum_{j=1}^{s} n. \quad (5)$$

In the worst case, $s = l$ and T becomes

$$T = n \times l + n \times l = 2(n \times l). \quad (6)$$

Thus, the worst case time complexity for algorithm *GreedyMatching* is $O(n \times l)$, where n is the total number of the animes in the graph, and l is the the length of the input search sequence. It is worth noting that a few of our worst case assumptions are actually impossible. Our experiments indicate that the average case time complexity is $O(\frac{n}{p} \times l)$ where p is the number of phonemes in the English language.

References

[AF02] ARIKAN O., FORSYTH D. A.: Interactive motion generation from examples. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), ACM Press, pp. 483–490. 2, 3

[BCS97] BREGLER C., COVELL M., SLANEY M.: Video rewrite: driving visual speech with audio. In *SIGGRAPH 97 Conference Proceedings* (Aug. 1997), ACM SIGGRAPH, pp. 353–360. 2

[Bra99] BRAND M.: Voice puppetry. In *Proceedings of ACM SIGGRAPH 1999* (1999), ACM Press/Addison-Wesley Publishing Co., pp. 21–28. 2

[BS94] BROOK N., SCOTT S.: Computer graphics animations of talking faces based on stochastic models. In *International Symposium on Speech, Image Processing, and Neural Networkds* (1994). 2

[Buh03] BUHMANN M. D.: *Radial Basis Functions : Theory and Implementations*. Cambridge University Press, 2003. 3

[CM93] COHEN N., MASSARO D. W.: Modeling coarticulation in synthetic visual speech. In *Models and Techniques in Computer Animation* (1993), Thalmann N. M., Thalmann D., (Eds.), Springer–Verlang, pp. 139–156. 2

[CPB*94] CASSELL J., PELACHAUD C., BADLER N., STEEDMAN M., ACHORN B., BECKET W., DOUVILLE B., PREVOST S., STONE M.: Animated conversation: Rule-based generation of facial expression, gesture and spoken intonation for multiple conversational agents. In *Proceedings of ACM SIGGRAPH 1994* (1994). 2

[CXH03] CHAI J., XIAO J., HODGINS J.: Vision-based control of 3d facial animation. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2003), Eurographics Association, pp. 193–206. 2

[EGP02] EZZAT T., GEIGER G., POGGIO T.: Trainable videorealistic speech animation. In *Proceedings of ACM SIGGRPAH 2002* (2002), ACM Press, pp. 388–398. 2

[Int] INTERNATIONAL COMPUTER SCIENCE INSTITUTE, BERKELEY, CA: Rasta software. www.icsi.berkeley.edu/Speech/rasta.html. 3

[KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. In *Proceedings of ACM SIGGRAPH 2002* (2002), ACM Press, pp. 473–482. 2

[KMG02] KALBERER G. A., MUELLER P., GOOL L. V.: Speech animation using viseme space. In *Vision, Modeling, and Visualization VMV 2002* (2002), Akademische Verlagsgesellschaft Aka GmbH, Berlin, pp. 463–470. 2

[LCR*02] LEE J., CHAI J., REITSMA P., HODGINS J., POLLARD N.: Interactive control of avatars animated with human motion data, 2002. 2, 3

[LTW95] LEE Y., TERZOPOULOS D., WATERS K.: Realistic modeling for facial animation. In *SIGGRAPH 95 Conference Proceedings* (Aug. 1995), ACM SIGGRAPH, pp. 55–62. 2

[LWS02] LI Y., WANG T., SHUM H.-Y.: Motion texture: A two-level statistical model for character motion synthesis. *ACM Transactions on Graphics 21*, 3 (July 2002), 465–472. 2, 3

[MKT*98] MASUKO T., KOBAYASHI T., TAMURA M., MASUBUCHI J., K. TOKUDA: Text-to-visual speech synthesis based on parameter generation from hmm. In *ICASSP* (1998). 2

[Pel91] PELACHAUD C.: *Realistic Face Animation for Speech*. PhD thesis, University of Pennsylvania, 1991. 2

[SBCS04] SAISAN P., BISSACCO A., CHIUSO A., SOATTO S.: Modeling and synthesis of facial motion driven by speech. In *European Conference on Computer Vision 2004* (2004), pp. 456–467. 2

[SG] SPEECH GROUP C. M. U.:. `www.speech.cs.cmu.edu/festival`. 3, 4

[Wat87] WATERS K.: A muscle model for animating three-dimensional facial expression. In *SIGGRAPH 87 Con-*

ference Proceedings) (July 1987), vol. 21, ACM SIGGRAPH, pp. 17–24. 2

Figure 4: *A long synthesized utterance using algorithm* GreedyMatching.