Morphing Spectral Envelopes Using Audio Flow

Tony Ezzat[†], *Ethan Meyers*[†], *Jim Glass*[‡]
Tomaso Poggio[†]

Center for Biological and Computational Learning[†] &
Computer Science and Artificial Intelligence Laboratory[‡]
Massachusetts Institute of Technology, Cambridge, MA
tonebone@mit.edu, emeyers@mit.edu, glass@mit.edu, tp@ai.mit.edu

Abstract

We present a method for morphing between smooth spectral magnitude envelopes of speech. An important element of our method is the notion of *audio flow*, which is inspired by similar notions of *optical flow* computed between images in computer vision applications. Audio flow defines the correspondence between two smooth spectral magnitude envelopes, and encodes the formant shifting that occurs from one sound to another. We present several algorithms for the automatic computation of audio flow from a small 20 second corpus of speech. In addition, we present an algorithm for morphing smoothly between any two spectral magnitude envelopes, given the computed audio flow between them.

1. Introduction

In various speech processing applications, there is a need to interpolate smoothly between two sounds. These applications include speech synthesis, where there is a need to reduce the spectral mismatch at concatenation join points; in speech compression, where there is a need to interpolate between two decoded spectra; and speech recognition, where there may be a need to interpolate between different people's phonemes for adaptation purposes.

We focus in this work on interpolating *smoothed spectral magnitude envelopes* [1] instead of complete sound spectra, deferring the problems of interpolating residuals to future work. Smooth spectral magnitude envelopes (see Figure 1) capture the formant resonances of the vocal tract, and, as such, different envelopes for different sounds will exhibit different formant locations, shapes, and amplitudes.

In this work, we present a *morphing algorithm* which morphs smoothly between any two spectral magnitude envelopes $S_1(f)$ and $S_2(f)$ of speech. In doing so, the morphing algorithm smoothly interpolates formant locations, shapes, and amplitudes between the two envelopes. As in the image case [2], morphing transformations between two envelopes occur as a *warp* of the first envelope into the second, a similar *inverse warp* of the second envelope into the first, and a final *cross-dissolve* or *blend* of the warped envelopes.

The difficulty with morphing approaches is that the specification of the warp requires establishing *correspondence* between the envelopes. Establishing correspondence ensures that the warping process preserves the desired alignment between the geometric attributes of the objects as they are morphed. For example, we would want the first formant in one envelope to map to the first formant in the other envelope, the second formant in one envelope to map to the second formant in the second envelope, and so on. Consequently, this correspondence between the envelopes needs to be specified.

In this work, we have resorted to *audio flow methods* to estimate correspondence. The notion of audio flow is inspired by the analogous notion of *optical flow* from computer vision. Optical flow, originally formulated by Horn & Schunck [3], measures the motion of objects between two images. Similarly, audio flow measures motion between two smoothed envelopes. Here we represent audio flow as a one-dimensional vector field $\Delta f(f)$ that describes how spectral energy at each frequency bin f has moved between two envelopes $S_1(f)$ and $S_2(f)$.

In Sections 4 through 7, we present several algorithms for audio flow computation, each one successively improving on the previous one and culminating in our final algorithm. The final audio flow algorithm is *automatic*, requiring no hand-labelling; it is *dense*, assigning correspondence for each frequency bin; it is *nonparametric*, requiring no parametric representation of the envelopes; finally, it is *data-driven*, estimating correspondence from natural transition paths extracted from a small 20-second data corpus. In section 8 we present our morphing algorithm which uses the computed audio flow.

2. Previous Work

The simplest way to interpolate two envelopes is to linearly cross-fade between them [4], but this technique does not account for proper formant shifting between two sounds. Linearly cross-fading *alternative representations* of the envelopes (such as line spectral frequencies [5]) also does not account for proper formant shifting (as was shown in [6] [7]).

Sinusoidal methods [8] utilize a nearest neighbor correspondence estimation procedure which tracks *partials* across a spectrogram. However, tracking partials is difficult due to their noisy nature, and partial movement is affected by pitch changes as well as formant changes.

The pole-shifting method of Goncharoff & Kaine-Krolak [9] attempts to establish correspondence between poles in an LPC representation of the envelopes, but pole-matching is a difficult process, and the relationship between poles and formants is not one-to-one.

The approach closest to ours is the Pfitzinger's dynamic frequency warping (DFW) approach [7], in which DFW is used to establish correspondence non-parametrically between two LPC envelopes, and a morphing algorithm is used to generate the intermediate envelopes. We defer our discussion of this approach until the last section of our paper.

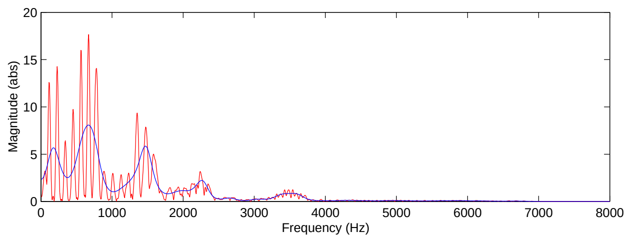

Figure 1: Smoothed spectral envelope $S_t(f)$ overlayed on top of its corresponding magnitude spectrum $\|R_t(f)\|$.

3. Smooth Spectral Magnitude Envelope Computation

Our spectral envelopes are obtained from a 20-second audio database consisting of 9 Harvard List sentences sampled at 16Khz. All of the sentences were STFT-analyzed using a 48msec Hamming window with a 2ms frame rate and a zeropadding factor of 4, yielding a total of 9000 3072-dimensional audio frames $\{R_t(f)\}_{t=1}^{9000}$ for the entire database.

For each frame, we extracted the smooth spectral magnitude envelope $S_t(f)$ using a *cepstral* method [1]: first, a discrete cosine transform (DCT) is performed on the log magnitude of the spectrum; the DCT coefficients are then truncated to 104 coefficients; finally, the DCT is inverted back into frequency. Figure 1 shows an example of a smoothed spectrum $S_t(f)$ overlayed on top of its corresponding original magnitude spectrum $\|R_t(f)\|$.

4. Direct Audio Flow

Given two smoothed spectral magnitude envelopes $S_1(f)$ and $S_2(f)$, the *direct audio flow* algorithm computes an audio flow estimate $\Delta f(f)$ *directly* from the spectral envelopes, without any recourse to other data.

In order to be able to estimate flow independent of any arbitrary multiplicative gains in the envelopes, we estimate flow from the *derivative of the log magnitudes*. More formally, we estimate flow between $\overline{S_1}(f)$ and $\overline{S_2}(f)$, where

$$\overline{S_1}(f) \doteq \frac{d(log_{10}S_1(f))}{df}, \quad \overline{S_2}(f) \doteq \frac{d(log_{10}S_2(f))}{df} \qquad (1)$$

To solve for $\Delta f(f)$, we first make the *smoothness* assumption that the audio flow is *locally similar* over a frequency region R, and formulate an error objective based on the l_2 norm:

$$E(f) = \sum_R (\overline{S_1}(f+\Delta f) - \overline{S_2}(f))^2 \qquad (2)$$

Furthermore, we make the assumption that the audio flow is *small*, allowing us to expand the first term on the right hand side above using a Taylor series and discard the higher order terms:

$$\overline{S_1}(f+\Delta f) = \overline{S_1}(f) + \overline{S_1}'(f)\Delta f \qquad (3)$$

Combining Equations 2 and 3, we can solve for an estimate of the flow at each point by taking the derivative and setting to zero:

$$\Delta f(f) = \frac{\sum_R \overline{S_1}'(f)(\overline{S_2}(f) - \overline{S_1}(f))}{\sum_R \overline{S_1}'(f)^2} \qquad (4)$$

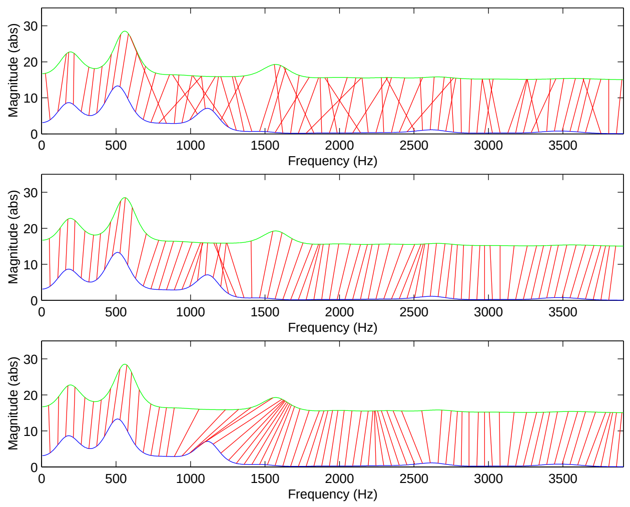

Figure 2: Two envelopes and the audio flow computed between them. Top: Flow computed using the direct flow algorithm. Middle: Flow computed using the coarse-to-fine flow algorithm. Bottom: Flow computed using concatenation along a path extracted from the corpus graph.

The above Equation is the famous Lucas-Kanade [10] equation which frames the audio flow in terms of the *frequency* derivative $\overline{S_1}'(f)$ and the *temporal* derivative $\overline{S_2}(f) - \overline{S_1}(f)$. In our case the width of region R is set to 5. In addition, the flow Δf is set to 0 if the denominator term $\sum_R \overline{S_1}'(f)^2$ is smaller than 10^{-5}. Equation 4 is solved for all 3072 frequency bins, although half as many values need to be computed since the envelope is symmetric.

Shown in the top part of Figure 2 is the audio flow estimated between two envelopes. Experimentally, we have found that in many cases the direct audio flow algorithm produces noisy and inaccurate motion estimates between various spectral magnitude envelopes, as may be seen for the case of the first and second formants in top part of the figure. This is because audio flow displacements between spectral magnitude envelopes can be significantly large, so the small motion assumption of Equation 3 is violated, and the direct audio flow algorithm fails.

5. Coarse-to-Fine Audio Flow

In order to overcome the problems with large motions between envelopes, it is possible to embed the audio flow estimation procedure in a multiscale pyramidal framework, in which each envelope is reduced in resolution using a Gaussian pyramid [11] [12]. Reducing the resolution reduces the magnitude of the motion that has occurred between the envelopes, preventing the small motion assumption of Equation 3 from being violated. Audio flow displacement estimates are initially obtained at coarse resolutions, and then propagated to higher resolution levels of the pyramid.

Our *coarse-to-fine audio flow* algorithm mirrors the algorithm presented by Bergen et al [11] for the image case. Included as a subroutine in the algorithm is the direct flow algorithm from Section 4. Typically we create a 4-level pyramid for the 3072-bin envelopes in our dataset. Pseudocode for our algorithm is presented on our web page [13].

Shown in the middle of Figure 2 is the result of the multiresolution flow estimation procedure. The multiresolution al-

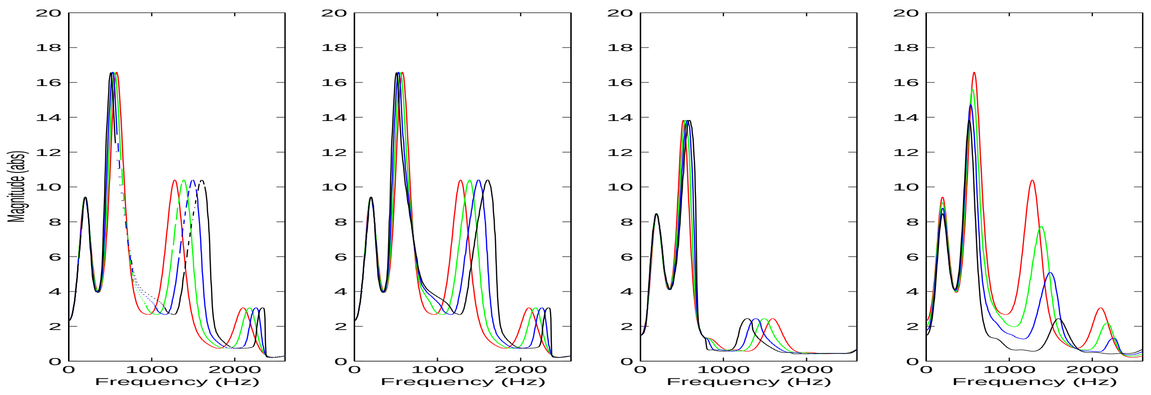

Figure 3: The process of morphing two envelopes with significant second formant movement. Far left: Forward warping $S_1(f)$ without hole-filling. Middle left: Forward warping $S_1(f)$ with hole-filling. Middle right: Forward warping $S_2(f)$ with hole-filling. Far right: Morphing $S_1(f)$ and $S_2(f)$.

gorithm produces smoother flow estimates than the direct flow algorithm. Despite this, the multiresolution algorithm is still not able to estimate reliable motion estimates when formant motions are very large. This is illustrated in Figure 2 for the second formant (which moves to the right).

6. Concatenated Audio Flow

In cases where the coarse-to-fine audio flow between two envelopes is not sufficient, we have found that a *concatenation* procedure improves estimates further. Since the 20-second audio corpus is analyzed at 500 fps, there are many intermediate frames that lie between any two chosen envelopes $S_1(f)$ and $S_2(f)$. A series of consecutive audio flow vectors between each intermediate frame and its successor may be computed and *concatenated* into one large audio flow vector that defines the global transformation between the chosen envelopes.

Our flow concatenation algorithm works as follows: Given a series of consecutive frames $S_0, S_1, \ldots S_n$, we would like to construct the audio flow $\Delta f_{0(n)}$ relating S_0 to S_n. We focus on the case of the 3 envelopes S_{i-1}, S_i, S_{i+1} since the concatenation algorithm is simply an iterative application of this 3-frame base case.

Audio flow is first computed between the consecutive frames to yield $\Delta f_{(i-1)i}$ and $\Delta f_{i(i+1)}$. Both flows are computed using the coarse-to-fine audio flow algorithm of Section 5. Then, to place all flows in the same reference frame, the audio flow $\Delta f_{i(i+1)}$ is *backwards warped* along $\Delta f_{(i-1)i}$ to create $\Delta f^{warped}_{i(i+1)}$. Finally, $\Delta f^{warped}_{i(i+1)}$ and $\Delta f_{(i-1)i}$ are both added to produce an approximation to the desired concatenated audio flow. Additional pseudocode for our concatenation procedure is provided in [13].

Concatenation produces good audio flow only when the number of frames between $S_1(f)$ and $S_2(f)$ are small in number. However, since any two envelopes are usually very far apart in the corpus, repeated concatenation that would be involved across the hundreds or thousands of intermediate frames leads to a considerably degraded final flow. A method is needed to figure out how to compute a *short* path from $S_1(f)$ to $S_2(f)$, to avoid repeated concatenation over hundreds or thousands of intermediates frames.

7. Audio Flow through a Graph

We obtain short paths between any two envelopes by constructing the *corpus graph* representation of the corpus: A corpus graph is an N-by-N sparse adjacency graph matrix in which each frame $\{S_t(f)\}_{t=1}^{N}$ in the corpus is represented as a node in a graph connected to k nearest frames. The k nearest frames are chosen using the *k-nearest neighbors* algorithm, and the distance metric d used is the log Euclidean distance:

$$d(S_1(f), S_2(f)) = \| log_{10} S_1(f) - log_{10} S_2(f) \| \qquad (5)$$

We set $k = 250$ in this work.

After the corpus graph is computed, the *Dijkstra* shortest path algorithm is used to compute the shortest path between the envelopes $S_1(f)$ and $S_2(f)$. Each shortest path produced by the Dijkstra algorithm is a list of envelopes from the corpus that cumulatively represent the shortest deformation path from $S_1(f)$ to $S_2(f)$ as measured by the log Euclidean distance. Concatenated flow from $S_1(f)$ to $S_2(f)$ is then computed along the intermediate envelopes as in Section 6.

Compared to all the methods described in Sections 4 through 6, we have found that computing audio flow through a corpus graph is our best method to establish correspondence between any two envelopes. Shown in the bottom of Figure 2 is the audio flow estimated between the two envelopes in the top using a 20-second corpus graph. As may be seen, the flow correctly estimates the difficult movement of the second formant.

8. Morphing

Given two envelopes $S_1(f)$ and $S_2(f)$, the first step of our morphing algorithm is to compute audio flow Δf_{12} between them using the algorithm presented in Section 7.

The second step of our algorithm is to *forward warp* $S_1(f)$ along Δf_{12}. Our forward warping algorithm “pushes” the values of $S_1(f)$ along the flow Δf_{12}. By scaling Δf_{12} uniformly by a constant α between 0 and 1, one can produce a series of warped intermediate frames which approximate the transformation between $S_1(f)$ and $S_2(f)$. Several such intermediate warps are shown in Figure 3 (far left). Notationally, we denote the forward warping operation as an operator $\mathbf{W}(S, \Delta f)$ that operates on an envelope S and audio flow Δf.

The *holes* which appear in the intermediate frames shown in Figure 3 (far left) occur in cases where a destination fre-

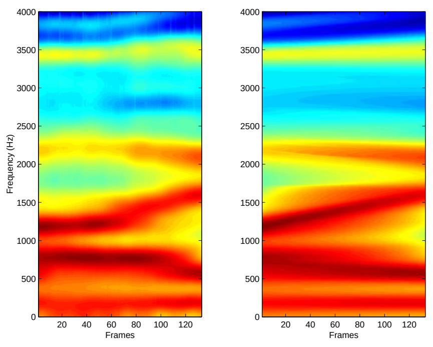

Figure 4: Real /ay/ transition spectrogram (left) along a morphed /ay/ transition spectrogram. The first and last frames of both sounds are identical.

quency bin was not filled in with any source bin value. The reason for this is that the forward warping algorithm rounds to the nearest integer when it decides which destination bin to fill, in addition to the fact the envelope transformations themselves exhibit nonzero divergence. We use a hole-filling algorithm that traverses the warped envelope in left-to-right order and fills in the holes by interpolating linearly between their non-hole endpoints. Figure 3 (middle left) shows the same set of warped intermediates as in Figure 3 (far left), but with the holes filled in using our algorithm.

Since the second warp in a morph is a warp of $S_2(f)$ towards $S_1(f)$, an *inverse flow* Δf_{21} from $S_2(f)$ to $S_1(f)$ needs to be computed. In this work, the inverse flow is computed using the same algorithm presented in Section 7. Figure 3 (middle right) depicts the set of envelopes generated as a result of warping $S_2(f)$ along the inverse flow from $S_2(f)$ to $S_1(f)$.

Finally, the morphing algorithm synthesizes an intermediate envelope $S_{morph}(\alpha)$ at position α by *cross-dissolving* or *blending* the warped intermediates:

$$S_{morph}(\alpha) = (1-\alpha)\mathbf{W}(S_1(f), \alpha\Delta f_{12}) + \alpha\mathbf{W}(S_2(f), (1-\alpha)\Delta f_{21}) \quad (6)$$

By interpolating the blending parameter α appropriately the morph "fades out" the warped versions of $S_1(f)$ and "fades in" the warped versions of $S_2(f)$. A final morph sequence is shown in Figure 3 (far right). The parameter α is interpolated *linearly* between 0.0 and 1.0. See [13] for more details on our morphing algorithm.

9. Results & Discussion & Future Work

A whole array of morphs from various envelopes in our corpus are presented in [13]. We have found that audio flow morphing is capable of successfully handling a wide variety of envelope phenomena such as formant shifts, amplitude changes, and formant appearances/disappearances. Experimentally, the audio flow algorithm presented in Section 7 works the most reliably.

Shown in Figure 4 is a real `/ay/` transition along with a morphed `/ay/` transition for comparison. As may be seen, our morphing algorithm is *linear*, in the sense that the formant trajectories follow linear paths, and bandwidths change linearly as well. As such, our algorithm does not account for the nonlinear trajectory and bandwidth changes which occur during sound transitions.

One important factor in algorithm performance is the size of the corpus: if the corpus is too small, there will not be enough data to generate smooth paths through the graph. Empirically we have found that a 20 second corpus is sufficiently large, although it sometimes produces incorrect results and needs to be enlarged (as may be seen in [13]).

Pfitzinger's DFW algorithm [7] also produces good audio flow estimates between envelopes. The difference between the algorithms is that the DFW algorithm is *direct*, making no recourse to any data, while our algorithm is *data-driven*, relying on a small corpus to extract natural transition paths between envelopes. Our data-driven approach yields more natural flow estimates in certain cases, although it is also possible to embed the DFW algorithm in a coarse-to-fine and data-driven framework.

Future work will focus on interpolating *complete* sounds rather than just envelopes, as well as evaluating the perceptual quality of the morphs.

10. References

[1] D. Schwarz and X. Rodet, "Spectral envelope estimation and representation for sound analysis-synthesis," in *Proceedings of ICMC*, Beijing, China, October 1999.

[2] T. Beier and S. Neely, "Feature-based image metamorphosis," in *Computer Graphics (Proceedings of ACM SIGGRAPH 92)*, vol. 26(2). Chicago, IL: ACM, 1992, pp. 35–42.

[3] B. K. P. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, pp. 185–203, 1981.

[4] M. C. M. Slaney and B. Lassiter, "Automatic audio morphing," in *Proc. ICASSP*, Atlanta Georgia, 1996.

[5] K. Paliwal, "Interpolation properties of linear prediction parametric representations," in *Proceedings of Eurospeech*, vol. 2, Madrid, 1995, pp. 1029–1032.

[6] D. T. Chappell and J. H. L. Hansen, "A comparison of smoothing methods for segment concatenation," *Speech Communication*, vol. 36, pp. 343–374, 2002.

[7] H. R. Pfitzinger, "Dfw-based spectral smoothing for concatenative speech synthesis," in *Proceedings of ICSLP*, vol. 2, Korea, October 2004, pp. 1397–1400.

[8] R. McAulay and T. Quatieri, "Speech analysis/synthesis based on a sinusoidal representation," *IEEE Trans. Acoustics, Speech and Signal Processing*, vol. Vol. ASSP-34, no. 4, pp. 744–754, August 1986.

[9] V. Goncharoff and M. Kaine-Krolak, "Interpolation of lpc spectra via pole-shifting," in *Proceedings of ICASSP*, vol. 1, Detroit, May 1995, pp. 780–783.

[10] B. Lucas and T. Kanade, "An iterative image registration technique with an application to stero vision," in *Proc. DARPA Image Understanding Workshop*, 1981, pp. 121–130.

[11] J. Bergen and R. Hingorani, "Hierarchical motion-based frame rate conversion," David Sarnoff Research Center, Princeton, New Jersey, Tech. Rep., Apr. 1990.

[12] J. Bergen, P. Anandan, K. Hanna, and R. Hingorani, "Hierarchical model-based motion estimation," in *Proceedings of the European Conference on Computer Vision*, Santa Margherita Ligure, Italy, 1992, pp. 237–252.

[13] [Online]. Available: http://cuneus.ai.mit.edu:8000/research/audioflow/