Implementation of sequential real-time waveform generator for high-quality vocoder

Masanori Morise*†
* School of Interdisciplinary Mathematical Sciences, Meiji University, Japan
† JST, PRESTO, Japan
E-mail: mmorise@meiji.ac.jp

***Abstract*—We describe an implementation of real-time waveform generation from vocoded speech parameters. High-quality vocoders such as STRAIGHT and WORLD have been used for voice conversion and statistical parametric speech synthesis. The current implementation of such vocoders has a function for generating the whole waveform from the speech parameters in all frames at one time. To sequentially generate a short-period waveform, implementations such as realtime STRAIGHT have been proposed. However, the generated speech waveform is inferior in sound quality to that of the original vocoder. To achieve sequential real-time waveform generation, a struct named *WorldSynthesizer* (WS struct) and six functions were implemented. The implementation is based on the WORLD vocoder, and it can generate the completely same waveform as the original except for the several points such as random seed used for generating the white noise. We therefore evaluated its processing speed by using the real time factor (RTF). The results showed that the processing speed of the proposed implementation decreased by 14.5% compared with the original WORLD. On the other hand, the RTF of the proposed implementation calculated from female speech was below 0.1, which suggests that the implementation is able to carry out real-time synthesis.**

I. Introduction

Vocoder-based high-quality speech analysis/synthesis systems have been used for various purposes, such as statistical parametric speech synthesis (SPSS) [1], [2], singing synthesis [3], and voice conversion [4]. In SPSS, although WaveNet [5] has been used for high-quality speech synthesis, the vocoder-based approach is still used for its flexibility. The high-quality vocoder on the basis of the channel vocoder [6] decomposes a speech waveform into the fundamental frequency (F0), spectral envelope, and aperiodicity. Although other speech parameters [7], [8], [9] have been proposed for similar purposes, the vocoder-based approach using three speech parameters is still popular.

Several high-quality vocoders, such as STRAIGHT [10], [11] and WORLD [12] (D4C edition [13]), have been used for SPSS. High-quality vocoders also have an algorithm for generating waveforms from the speech parameters. Their implementations require a complete set of speech parameters and generate the whole waveform from them at one time. When the speech parameters are sequentially generated by another system, the current waveform generator must wait for the end of the generation process. A real-time melody design interface such as v.morish [14] requires such a waveform generator. Sequential real-time waveform generation from sequential speech parameters would thus be useful.

Realtime STRAIGHT [15] is a means of real-time waveform generation. Although this system performs speech analysis/synthesis in real-time, its sound quality is inferior to that of the original STRAIGHT. For example, high-performance F0 estimators [16], [17] are used in high-quality vocoders, and their performance is superior to other traditional estimators such as Cepstrum [18], [19], YIN [20], and SWIPE [21]. However, since algorithms employed in the STRAIGHT and WORLD use both past and future information of the waveform, it is impossible for it to work in real time where there is no access to future information.

SPSS and real-time melody design interfaces use other frameworks to generate speech parameters. Since they don't require real-time speech analysis, only real-time synthesis is useful for them. Here, we attempted to implement a real-time waveform generator having compatibility with the original vocoder. We focused on the WORLD vocoder, a high-quality vocoder used all over the world. The purpose of this study was to develop a real-time waveform generator that would produce the same waveform as the original.

In Section 2 of this paper, we discuss related work on the waveform generation and describe the concept of the proposed implementation. In Section 3, we explain the details of the implementation. In Section 4, we evaluated the processing speed of the implementation in terms of the real time factor (RTF) and discuss the results. We conclude in Section 5 with a summary and a mention of future work.

II. Related work and concept of the proposed implementation

Several waveform generators that use vocoder-based speech parameters have been proposed. In this section, we review them and explain the baseline of the proposed implementation. We first explain them as related works and then explain the baseline for the proposed implementation. The requirements of the proposed implementation are also discussed.

A. Related work on waveform generation

STRAIGHT and WORLD use the same algorithm for generating waveforms. Other waveform generators, such as the log domain pulse model [22] and Vocaine [23], use different speech parameters from those of STRAIGHT and WORLD. In particular, the log domain pulse model uses the

phase distortion deviation (PDD) [24] and noise mask as its speech parameters. The Vocaine is similar to the sinusoidal representation [9]. In this research, we focus on high-quality vocoder based on channel vocoder because it has been used for various kinds of purposes.

Neural vocoders such as the WaveNet vocoder [25] have achieved high-quality speech synthesis. Their sound quality is better than that of conventional vocoders that don't use neural networks. However, since they require a high-performance graphics processing unit (GPU), they can't be used for real-time processing on a typical laptop PC. For this reason, we focused on an algorithm that does not require a GPU and devised an implementation that can achieve real-time synthesis on a laptop PC without a GPU.

B. WORLD: Baseline system

We used the WORLD vocoder as the baseline system. WORLD has several implementations in speech analysis; we employed the latest version consisting of Harvest [17], CheapTrick [26], [27], and D4C [13] as the F0, spectral envelope, and aperiodicity estimators, respectively. We expanded the waveform generation algorithm while ensuring that it would output the same waveform.

The original algorithm first calculates a parameter $\theta(t)$ from the F0 contour $F_0(t)$ to determine the temporal positions of the vocal cord vibrations.

$$\theta(t) = 2\pi \int_0^t F_0(\tau) d\tau, \tag{1}$$

Since the unvoiced frame has no F0 information, 500 Hz is set for all unvoiced frames as the default value in WORLD. The temporal position of the first vocal cord vibration is 0 s (origin), and the next position τ_1 is determined from $\theta(t)$. τ_1 is a value that satisfies the equation $\theta(\tau_1) - \theta(0) = 2\pi$. The n-th position $\theta(\tau_n)$ is recursively determined to satisfy the equation $\theta(\tau_n) - \theta(\tau_{n-1}) = 2\pi$.

The impulse response of a vocal cord vibration is calculated as the minimum phase response obtained from the spectral envelope. Pitch synchronous overlap and add (PSOLA) [28] is then carried out by using the impulse response. Since the speech waveform consists of not only the vocal cord vibration but also an aperiodic component, white noise is used as the excitation signal. In WORLD, the periodic and aperiodic components are independently calculated and then summed.

III. Implementation of Real-Time Waveform Generator

Here, we explain our implementation of real-time waveform generation from speech parameters. The source code is in C++ and has been released[1]. The implementation uses a struct called *WorldSynthesizer* (WS struct) and consists of six functions.

[1]https://github.com/mmorise/World

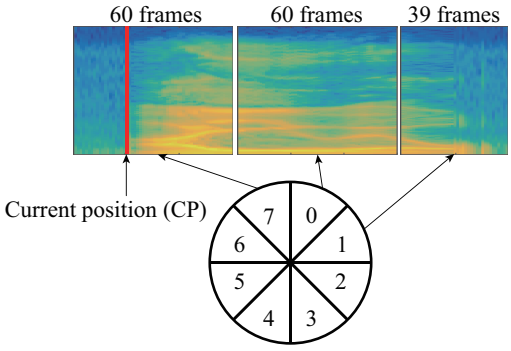

Fig. 1. Ring buffer in WS struct. The user can obtain the waveform from the current position (CP) to CP + N − 1th sample by using the Synthesis2 function.

A. Specifications of WS Struct

First, we explain real-time waveform generation using WS struct. WS struct has a ring buffer as shown in Fig. 1, and each buffer has a pointer to the speech parameters. The user can add several frames at one time. The current position (CP) in the figure is a member variable in WS struct, and the user can generate the waveform of N samples from this position. After that, the CP is automatically incremented by N samples. When the CP shifts to a temporal position in the next frame, the link in the current buffer is automatically removed from the ring buffer.

Sequential real-time waveform generation is performed as follows.

1) Initialization of WS struct by using the *InitializeSynthesizer* function.
2) Addition of speech parameters to WS struct by using the *AddParameters* function.
3) Generation of waveform (N sample) by using the *Synthesize2* function.
4) Release of the memory by using the *DestroySynthesizer* function.

If the user adds all frames at one time, he/she can call the Synthesis2 function until the current position reaches the end of the frame. The user can also add a speech parameter and call the Synthesis2 function alternately.

B. InitializeSynthesizer(): Initialization of WS struct

This function is called to set the following parameters of WS struct.

- Sampling frequency
- Number of samples N of the waveform generated by the Synthesis2 function
- Number of buffers in the ring buffer
- Frameshift length
- FFT length

Once WS struct is initialized, these parameters are fixed; the user cannot change them after they have been initialized. If a different condition is required, the user must build another WS struct with different parameters.

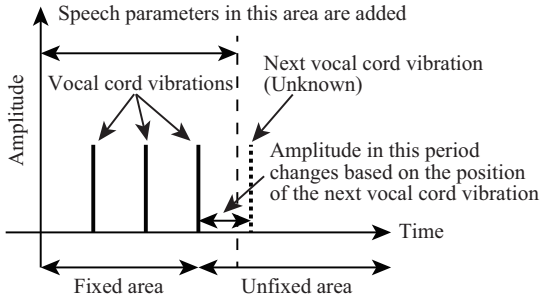

Fig. 2. The number of frames in speech parameters is different from the fixed area in which the user can generate the waveform.

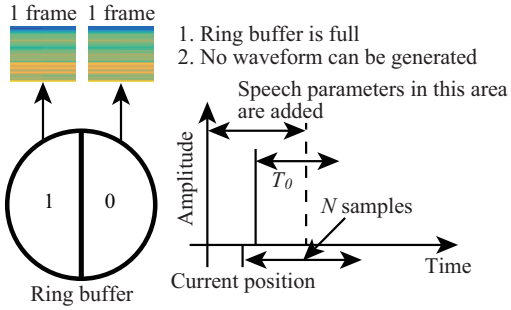

Fig. 3. WS struct is locked when the ring buffer is full, and the Synthesis2 function can output no waveform.

C. AddParameter(): Addition of speech parameters to WS struct

After initialization, the user can add speech parameters to the ring buffer. The unique aspect of this implementation is that the user can add an arbitrary number of frames at one time. Fig. 1 shows cases in which speech parameters with 60, 60, and 39 frames are added. If the ring buffer is full, the function will return an error message, and no speech parameter will be added. After calling this function, the number of times that the user can call Synthesis2 function is automatically updated.

D. Synthesis2(): Waveform generation for N samples

This function first judges whether the waveform of N samples can be generated from the CP and the linked speech parameters. As shown in Fig. 2, the waveform that can be generated is determined based on the added speech parameters. When the last vocal cord vibration is obtained at sample n, Synthesis2 function can output the waveform until sample $n-1$. When temporal positions of the vocal cord vibrations are included from the CP to CP+N, the function automatically calculates the vocal cord vibrations. Therefore, the processing speed of the Synthesis2 function depends on the number of vocal cord vibrations in the period.

E. IsLock(): Check of the status of WS struct

In the proposed implementation, the user sets the number of pointers in the ring buffer during the initialization. As shown in Fig. 3, no waveform can be generated even if the ring buffer is full. We call this status *locked*. Once WS struct is locked, the user must refresh it by using the *RefreshSynthesizer* function or clear the memory by using the DestorySynthesizer function. The IsLock function can be used to determine whether WS struct is locked.

This problem can be avoided by appropriately determining the number of pointers in the ring buffer and the number of frames in the AddParameter function. The user should guarantee that the product of the frameshift and the number of frames in the ring buffer always exceeds the longest fundamental period T_0.

F. DestorySynthesizer(): Release of WS struct

Since we used the C++ language to have compatibility with C, we must release the memory in WS struct after the processing. This function releases all memories used in WS struct. When the user has to reset WS struct to escape from a locked status, the RefreshSynthesizer function can be used instead of the DestorySynthesizer function.

IV. Evaluation

The proposed implementation can generate the completely same waveform as the original except for the several points such as random seed used for generating the white noise). Since the sound quality of the synthesized speech is the same, we did not evaluate it. Only the processing speed of the proposed implementation was compared with that of the original WORLD vocoder.

In the waveform generation, the number of FFTs is the most dominant parameter affecting the processing speed. It depends on the number of vocal cord vibrations, which means that the processing speed is proportional to F0. In cases where the same speech parameters are used for the comparison, the processing speed would be the same. The purpose of this evaluation was to verify the overhead of using WS struct in the proposed implementation.

A. Evaluation conditions

The processing speed evaluation used female speech and F0-modified speech. The sampling frequency was 48 kHz. Fig. 4 illustrates an example of an F0 contour estimated using Harvest [17]. The minimum and maximum values were 204 and 346 Hz, respectively. The evaluation used a laptop PC (i7-7500U 2.7 GHz, 16 GB memory).

In the initialization of WS struct, the number of samples in one processing was set to 256 (around 5.33 ms). The number of buffers in the ring buffer was 10, and the frameshift was 5 ms. The FFT length for calculating the spectral envelope and the aperiodicity was set to 2,048 samples. The RTF (real time factor) was used as the evaluation index, and the proposed implementation was compared with the WORLD vocoder implemented in C++. The histogram of the processing speed per processing was also calculated.

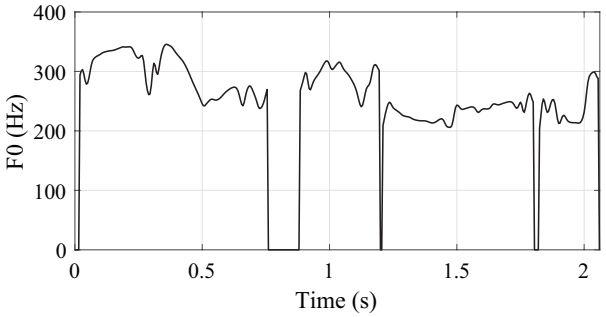

Fig. 4. Example of the F0 contour used in the evaluation.

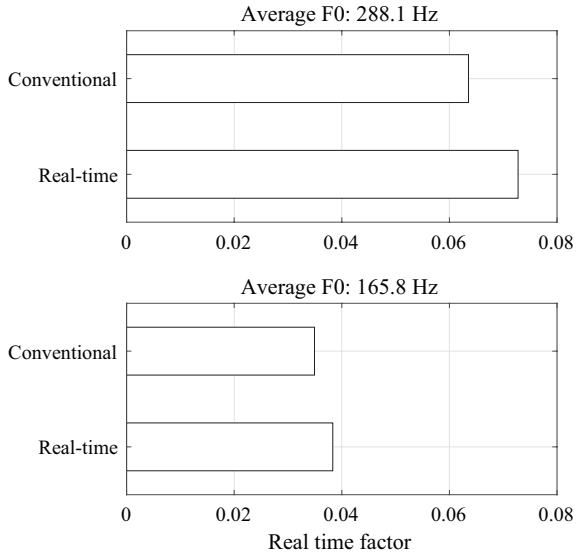

Fig. 5. Results of the evaluation.

B. Processing speed of generation of the whole waveform

First, we examine the time required to generate the waveform. The RTF was calculated 1,000 times, and the median value was used in the evaluation. Fig. 5 shows the results. The horizontal axis represents the RTF. The top represents the result of using the F0 contour of Fig. 4, and the bottom represents the result of using half of this F0 contour. Since the F0 value in the unvoiced frames is fixed to 500 Hz, the average of the bottom does not equal half the average of the top.

The proposed implementation was slower than the conventional algorithm by 14.5% on the top result. At the bottom, the difference was 9.7%. Because the RTF was below 0.1, we concluded that the overhead of the proposed implementation is small enough for it to generate waveforms in real time.

C. Processing speed of one processing of the Synthesis2 function

Next, we examine the processing speed per processing. Fig. 6 illustrates the histogram of the RTF. In this case, the RTF is calculated as the ratio between the processing speed in one processing and the 5.33 ms set as the parameter in WS struct. There are two peaks, and they are approximately related to the number of FFTs. The relationship between the number of FFTs and the RTF is discussed in the next section.

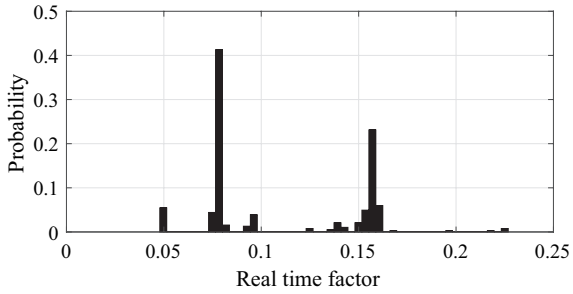

Fig. 6. Histogram of RTF in single processing of the Synthesis2 function.

The Synthesis2 function generates a waveform with a length of 5.33 ms. Since the lowest F0 is 204 Hz from Fig. 4, at least one vocal cord vibration is included in one processing. In WORLD and the proposed implementation, 500 Hz (2 ms) is set for the unvoiced frames. Up to three vocal cord vibrations are therefore included in one processing.

D. Discussion

The number of FFTs in one processing is calculated from the F0 information. We count the FFT and the inverse FFT as the same processing because their computational costs are almost the same. In cases where the frame has an F0 value, both voiced and unvoiced responses are calculated. Only the unvoiced response is calculated in the unvoiced frame.

Three (four) FFTs are required to generate the voiced (unvoiced) responses. To generate the voiced response, two FFTs are required to calculate the minimum phase spectrum from the spectral envelope. After that, the minimum phase response is calculated as the voiced response by using one FFT. Two FFTs are required to generate the unvoiced response, the same number as used to generate the voiced response. Finally, since the noise is used as the excitation signal, another FFT is required to calculate the spectrum of the noise. Therefore, a total of seven FFTs are required for the voiced section, while four FFTs in total are required for the unvoiced section.

We analyzed the relationship between the number of FFTs and the RTF by using the scatter plot shown in Fig. 7. The horizontal and vertical axes represent the number of FFTs and the RTF, respectively. The correlation coefficient between them was 0.998, which suggests that the RTF can be estimated using the number of FFTs. When the sample N was short, a high RTF was often observed. A long N is required to flatten the number of times of FFT. Since a large value of N causes a long latency, the user should take care of the balance between the RTF and N.

V. Conclusion

This paper described an implementation of a real-time waveform generator from vocoded speech parameters. The

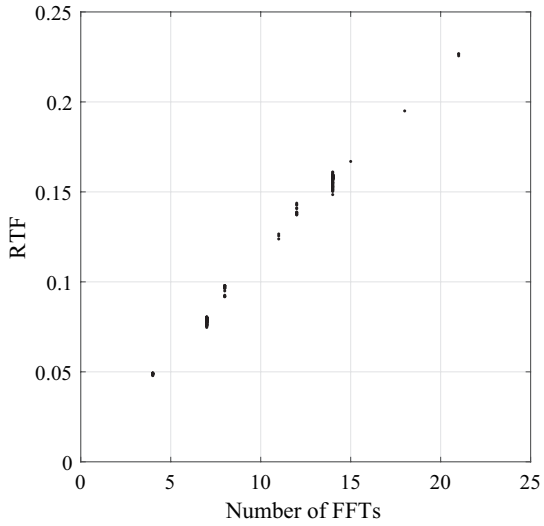

Fig. 7. Scatter plot of relationship between the number of FFTs and the RTF. The correlation coefficient was 0.998.

implementation is based on the WORLD vocoder, and it can generate almost the same waveform. The source code has been released via GitHub under a modified-BSD license. An evaluation showed that the implementation decreased the RTF by 14.5% compared with the original WORLD. However, the RTF remained under 0.1 in the synthesis of female speech. Since the processing speed is approximately proportional to F0, this result indicates the implementation is capable of carrying out sequential real-time synthesis.

The next step is to implement a real-time analysis system. Given such a system, users will be able to develop real-time voice conversion systems. It could be used for recording speech in a typical room. The development of a speech analysis algorithm robust against noise will also be an important work.

VI. Acknowledgements

This work was supported by a JST PRESTO Grant Number JPMJPR18J8.

References

[1] H. Zen, K. Tokuda, and A. W. Black, "Statistical parametric speech synthesis," *Speech Communication*, vol. 51, pp. 1039–1064, 2009.

[2] H. Zen, A. Senior, and M. Schuster, "Statistical parametric speech synthesis using deep neural networks," *in Proc. ICASSP2013*, pp. 7962–7966, 2013.

[3] M. Blaauw and J. Bonada, "A neural parametric singing synthesizer modeling timbre and expression from natural songs," *Applied Science*, vol. 7, no. 12, pp. 23–page, 2018.

[4] Y. Ohtani, T. Toda, H. Saruwatari, and K. Shikano, "Maximum likelihood voice conversion based on GMM with STRAIGHT mixed excitation," *in Proc. ICSLP*, pp. 2266–2269, 2006.

[5] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[6] H. Dudley, "Remaking speech," *J. Acoust. Soc. Am.*, vol. 11, no. 2, pp. 169–177, 1939.

[7] M. Airaksinen, B. Bollepalli, L. Juvela, Z. Wu, S. King, and P. Alku, "GlottDNN — a full-band glottal vocoder for statistical parametric speech synthesis," *in Proc. INTERSPEECH2016*, pp. 2473–2477, 2016.

[8] J. L. Flanagan and R. M. Golden, "Phase vocoder," *The Bell System Technical Journal*, vol. 45, no. 9, pp. 1493–1509, 2009.

[9] R. J. McAulay and T. F. Quatieri, "Speech processing based on a sinusoidal model," *The Lincoln Laboratory Journal*, vol. 1, no. 2, pp. 153–168, 1988.

[10] H. Kawahara, I. Masuda-Katsuse, and A. de Cheveigné, "Restructuring speech representations using a pitch-adaptive timefrequency smoothing and an instantaneous-frequency-based F0 extraction," *Speech Communication*, vol. 27, no. 3–4, pp. 187–207, 1999.

[11] H. Kawahara and M. Morise, "Technical foundations of TANDEM-STRAIGHT, a speech analysis, modification and synthesis framework," *SADHANA - Academy Proceedings in Engineering Sciences*, vol. 36, no. 5, pp. 713–728, 2011.

[12] M. Morise, F. Yokomori, and K. Ozawa, "WORLD: a vocoder-based high-quality speech synthesis system for real-time applications," *IEICE Trans. Inf. & Syst.*, vol. E99-D, pp. 1877–1884, 2016.

[13] M. Morise, "D4C, a band-aperiodicity estimator for high-quality speech synthesis," *Speech Communication*, vol. 84, pp. 57–65, 2016.

[14] M. Morise, M. Onishi, H. Kawahara, and H. Katayose, "v.morish'09: A morphing-based singing design interface for vocal melodies," *Lecture Notes in Computer Science*, vol. LNCS 5709, pp. 185–190, 2009.

[15] H. Banno, H. Hata, M. Morise, T. Takahashi, T. Irino, and H. Kawahara, "Implementation of realtime straight speech manipulation system," *Acoust. Science & Technology*, vol. 28, no. 3, pp. 140–146, 2007.

[16] H. Kawahara, A. Cheveigné, H. Banno, T. Takahashi, and T. Irino, "Nearly defect-free f0 trajectory extraction for expressive speech modifications based on straight," *in Proc. Interspeech2005*, pp. 537–540, 2005.

[17] M. Morise, "Harvest: A high-performance fundamental frequency estimator from speech signals," *in Proc. INTERSPEECH2017*, pp. 2321–2325, 2017.

[18] A. Noll, "Short-time spectrum and "cepstrum" techniques for vocal pitch detection," *J. Acoust. Soc. Am.*, vol. 36, no. 2, pp. 269–302, 1964.

[19] ——, "Cepstrum pitch determination," *J. Acoust. Soc. Am.*, vol. 41, no. 2, pp. 293–309, 1967.

[20] A. Cheveigné and H. Kawahara, "YIN, a fundamental frequency estimator for speech and music," *J. Acoust. Soc. Am.*, vol. 111, no. 4, pp. 1917–1930, 2002.

[21] A. Camacho and J. G. Harris, "A sawtooth waveform inspired pitch estimator for speech and music," *J. Acoust. Soc. Am.*, vol. 124, no. 3, pp. 1638–1652, 2008.

[22] G. Degottex, P. Lanchantin, and M. Gales, "A log domain pulse model for parametric speech synthesis," *IEEE/ACM Transactions on audio, speech, and language processing*, vol. 26, no. 1, pp. 57–70, 2018.

[23] Y. Agiomyrgiannakis, "Vocaine the vocoder and applications in speech synthesis," *in Proc. ICASSP 2015*, pp. 4230–4234, 2015.

[24] G. Degottex and D. Erro, "A uniform phase representation for the harmonic model in speech synthesis applications," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2014, no. 38, 2014.

[25] K. Kobayashi, T. Hayashi, A. Tamamori, and T. Toda, "Statistical voice conversion with WaveNet-based waveform generation," *in Proc. INTERSPEECH 2017*, pp. 1138–1142, 2017.

[26] M. Morise, "CheapTrick, a spectral envelope estimator for high-quality speech synthesis," *Speech Communication*, vol. 67, pp. 1–7, 2015.

[27] ——, "Error evaluation of an f0-adaptive spectral envelope estimator in robustness against the additive noise and f0 error," *IEICE Trans. Inf. & Syst.*, vol. E98-D, no. 7, pp. 1405–1408, 2015.

[28] E. Moulines and F. Charpentier, "Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones," *Speech Communication*, vol. 9, no. 5-6, pp. 453–467, 1990.