Adam Opsata
Lab Report 2. Digital Audio Systems 2013

World Around: a content dependant audio algorithm, for mobile audio devices.

Music is largely determined by its context (Byrne, 2011). In the 9th and 10th centuries Gregorian chant became a product the highly reverberant churches built in that era. During mid 20th century the three-and-a-half-minute pop song evolved in order to adapt to the playing time of a jukebox filled with vinyl 45 records. A study published in May 2011 found that "60.7 of all people listen to music on their mobile phones every day, while another 34.6 do so every now and then" (www.gsmarena.com, 2011), but the potential of these devices is hardly harnessed for music listening. Music has not yet adapted to today's most common venue.

The submitted function "World Around" is an algorithm designed to be implemented on a mobile computing device with a microphone, such as a smartphone or mp3 player. It allows a listener to hear recorded music that is modified by the sounds currently happening around them. Every time it is run it synthesizes a unique musical signal by using the pitches (phase information) from a recording provided by a composer, and the timbres (magnitude information) from the sounds currently happening in listener's environment. The function then combines this signal with an 'accompaniment' (the parts of the song the composer does not want effected by the listeners environment) and plays the resulting signal for the listener. The function works well in loud environments and has settings to compensate for quiet listening environments in different ways.

World Around solves other problems associated with listening to music on mobile devices. Headphones can be isolating. The listener may enjoy the diversion of music, but may not want to be cut off from the world around them. While using headphones one may be accidently impolite to someone they can't hear, or ignore a friend calling their name. They may be unaware of some danger that they could otherwise hear, like traffic at a crosswalk. Though it is not as effective as the unencumbered ear, the function does mitigate the cultural and safety issues of using headphones in public by giving the listener a greater degree of awareness of their surroundings. Furthermore, the world is full of beautiful sounds: birds chirping, children playing, and waves crashing. Even a city makes interesting hums, rattles and booms. A conventional music player deprives the listener of these sounds but World Around integrates their qualities into music.

A listener may have a favorite album that they have played too many times and therefore no longer listen to. Since World Around changes in every sonic surrounding, the lifespan of any recording produced for it is elongated. World Around is a new experience at every listen. Even on a first listen World Around creates new and unusual sounds that will hold ones attention.

Using World Around, identifying its signals, and adjusting its settings.

This function is intended to have 2 users, first a composer and then a listener. The composer provides the signals 'input_signal' and 'accompaniment' within World Around's code, and also specifies World Around's settings. The listener simply runs the function (a script is provided; just press run, and make sure there is some noise around you). The signal flow of World Around is illustrated in a block diagram provided in the figure below.

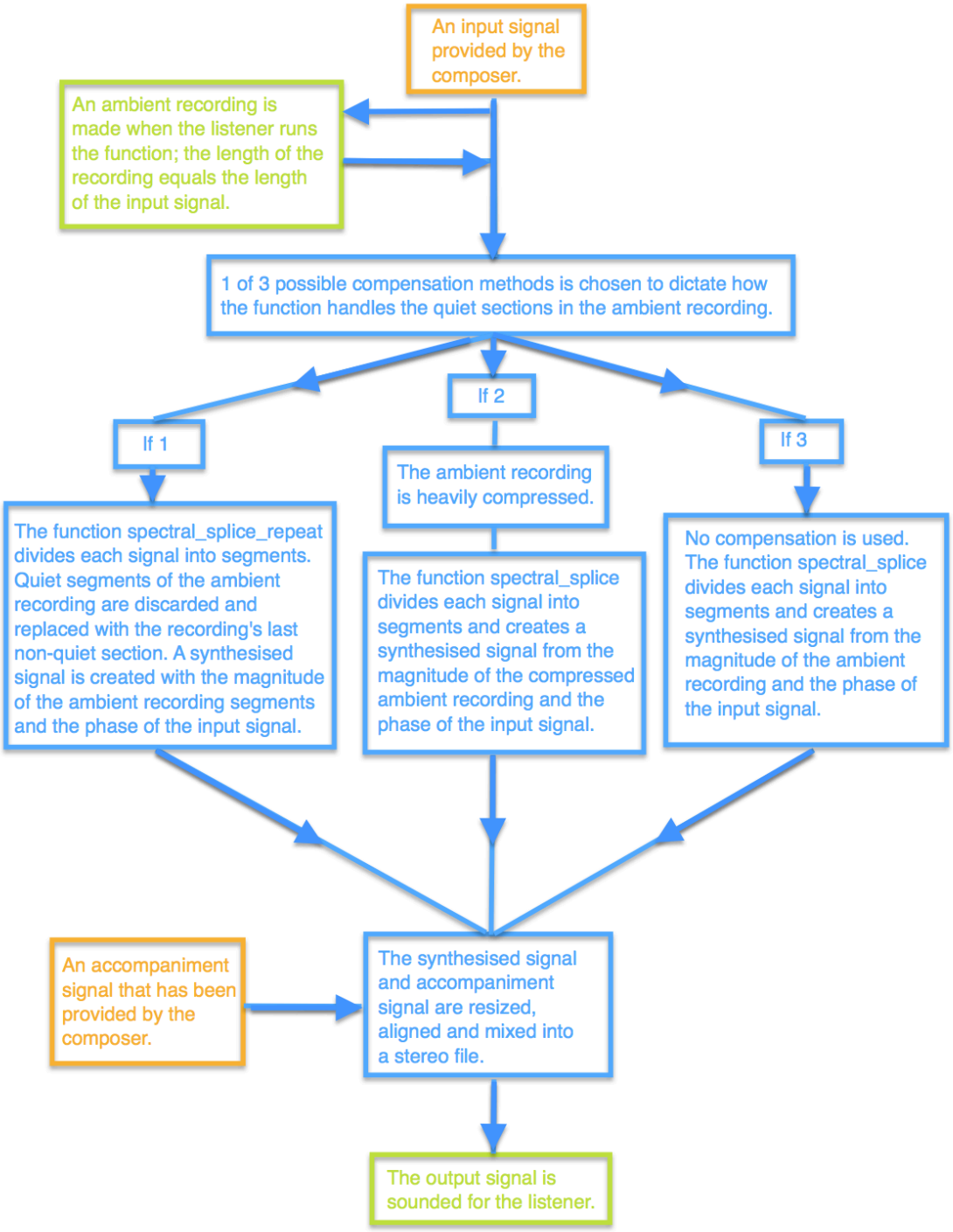
World_around produces a synthesized_signal using the magnitude of an input_signal and the phase of a second signal. The input_signal can either be a mono 44.1kHz .wav file, or a MIDI file. The function creates the second signal by recording the sound occurring in the listener's environment while the function is running. This second signal is called the ambient_recording.

As its final step World Around sounds an output_signal by combing the synthesized_signal with second signal provided by the composer called the 'accompaniment' signal. The composer provides a value for the time_in variable, which specifies the point in the accompaniment_signal when the synthesized_signal should begin playing, expressed in terms of its sample number. For example, to begin playing the synthesized_signal one second into the accompaniment_signal a value of 44100 should be used. If the two signals begin simultaneously a sample number of 0 should be used.

The synthesized_signal is produced by calling either the function spectral_splice_repeat, or spectral_splice. The mechanics of these functions are discussed in detail later. The benefits in choosing between these two are dependent on the dynamic range of the ambient_recording, and discussed below.

The ambient_recording is likely to have quiet sections in it, which can result in a boring synthesized_signal. The function's quiet_compensation setting can adapt to this in one of three ways. When quiet_compensation is set to 1 world_around calls the function spectral_splice_repeat. The function breaks each signal into small segments for synthesis. If a segment of the ambient_recording does not exceed a set RMS threshold it is replaced with the last segment that exceeded that threshold. This setting is suited to environments that jump between two volumes volume levels, for example a conversation in a quiet room. When quiet_compensation is set to 2 the ambient_recording is heavily compressed in order to increase the level of the ambient_recording's quiet sections. The compression is so heavy that distortion is also added to the ambient_recording, which has the benefit of adding colour to the synthesized_signal and noise to its quiet sections. This compressed signal is sent to the function spectral_splice, which is identical to spectral_splice_repeat except it does not reuse any segments. When quiet_compensation is set to 3 the ambient_recording is sent to spectral_splice with no compensation for volume level other than normalising. As a result a quiet portion of the ambient_recording results in a quiet portion of the synthesized_signal. This is the least robust setting of the function but it could be implemented for music that would not rely on the synthesized_signal. All settings do well in moderate to loud environments. Setting 1 is my favorite and the most robust.

A Block Diagram of World Around

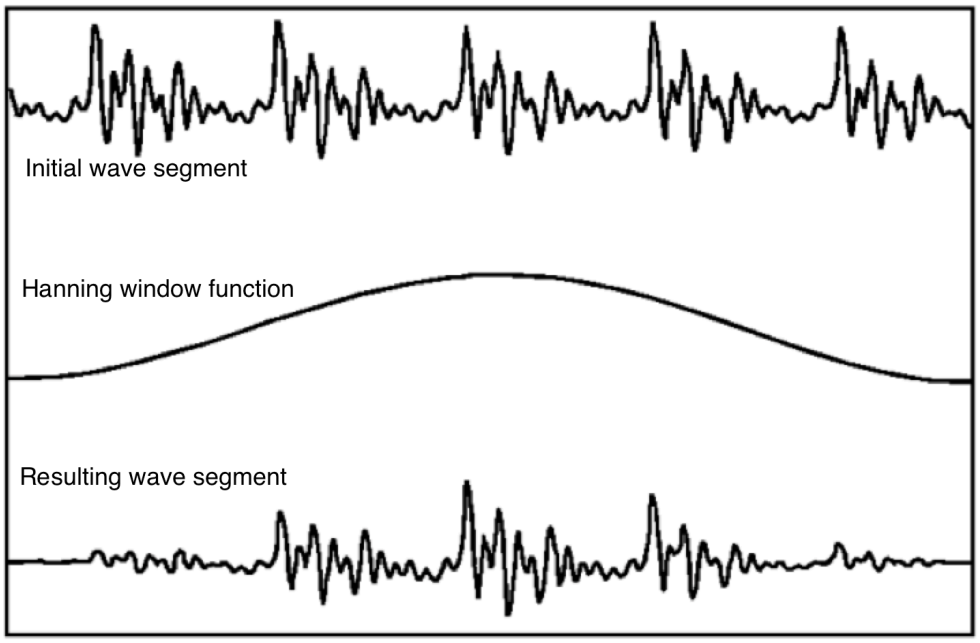


Spectral_splice, spectral_splice_repeat, and the math behind them.

Both of these functions have the input_signal and ambient_recording as inputs, and the synthesized_signal as an output. They both create a signal with the magnitude spectrum of one signal and the phase spectrum of the other, they only very in how they treat quiet portions of the signal.

The functions copy small segments from each input. The segment length is defined by the input 'window_size.' Each segment is multiplied by a hanning window. This decreases the segment's amplitude near its edges, which avoids the imprecision of analyzing the sharp edges of the segments.

Illustrating the Effect of a Hanning Window:



Values of the Hanning window approach zero as it nears its edges. When a wave is multiplied a Hanning window its values also approach zero near it's edges.

Image modified from: (www.mq.edu.au/acoustics, 2008)

Next the function transforms a segment from each signal into the time domain. It executes the complex discrete Fourier transform (DFT) by using Matlab's built-in fft function, which relies on the equation:

$$X[k] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]\, e^{-j2\pi kn/N}$$

where x[n] is the signal in the time domain , X[k], is the signal in the frequency domain. N is the number of samples in this analysis window, both n and k run from 0 to N-1, and j is the square root of -1.

Fourier analysis is founded on the fact that any complex wave can made by adding many pure

(sine and cosine) waves. The above equation returns an array of complex numbers that describes a series of pure waves. Each real number represents a cosine wave and each imaginary number represents a sine wave. The pure waves in the array, when added together, equal the wave in the segment that the function is currently analyzing. Each number in the array can be represented by z in the equation

$$z = x + \mathrm{i}\, y,$$

where x is the real value and y is the imaginary value, or values of x are the cosine waves signal is comprised of and y are its sine waves. For further illustration each point z can be plotted the complex plane.

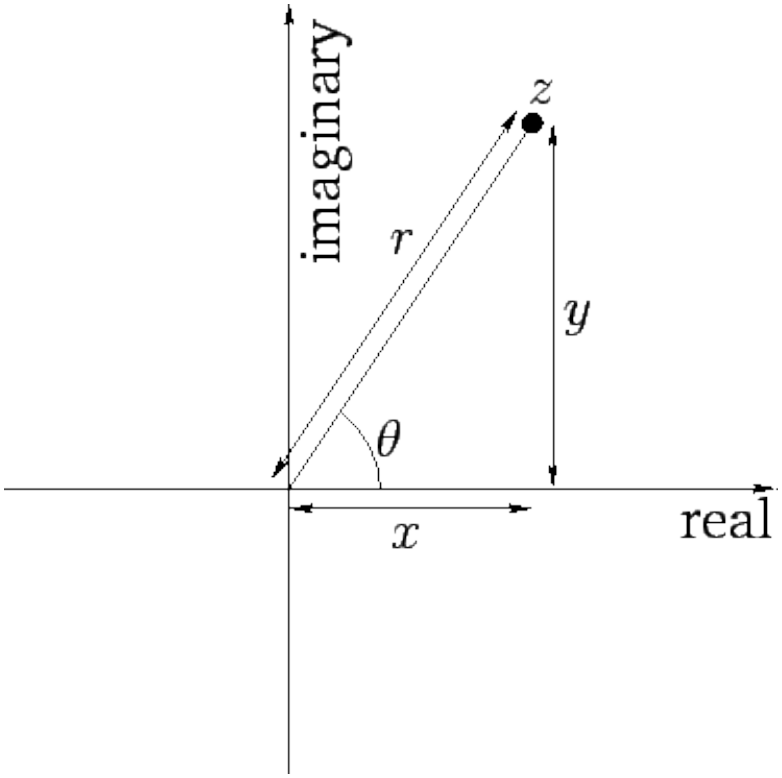


image source: (www.farside.ph.utexas.edu, 2010)

The magnitude of each wave is equal to its distance from the origin. This distance can be found using the Pythagorean theorem

$$r = \sqrt{x^2 + y^2}$$

where r is the distance from the origin, x is the real number and y is the imaginary number. When z is complex the Pythagorean theorem is mathematically identical finding z's absolute value. In this manner the functions spectral_splice, and spectral_splice_repeat obtain the

magnitude spectrum of each segment from the ambient_recording by finding the absolute value of its components after transforming it to the frequency domain.

The phase of each component wave can be obtained by finding z's angle from the real axis, illustrated as theta in the figure above. Basic geometry is once again employed to find theta by using the equation

$$\theta = \tan^{-1}(y/x)$$

where theta is the angle from the real axis, x is the real component of z and y is the imaginary component of z. The functions use this formula to obtain the phase of each pure wave in each segment of the input_signal.

After calculating the magnitude spectrum of the ambient_recording_segment and the phase spectrum of the input_signal_segment they are combined in to a single complex array, using the equation

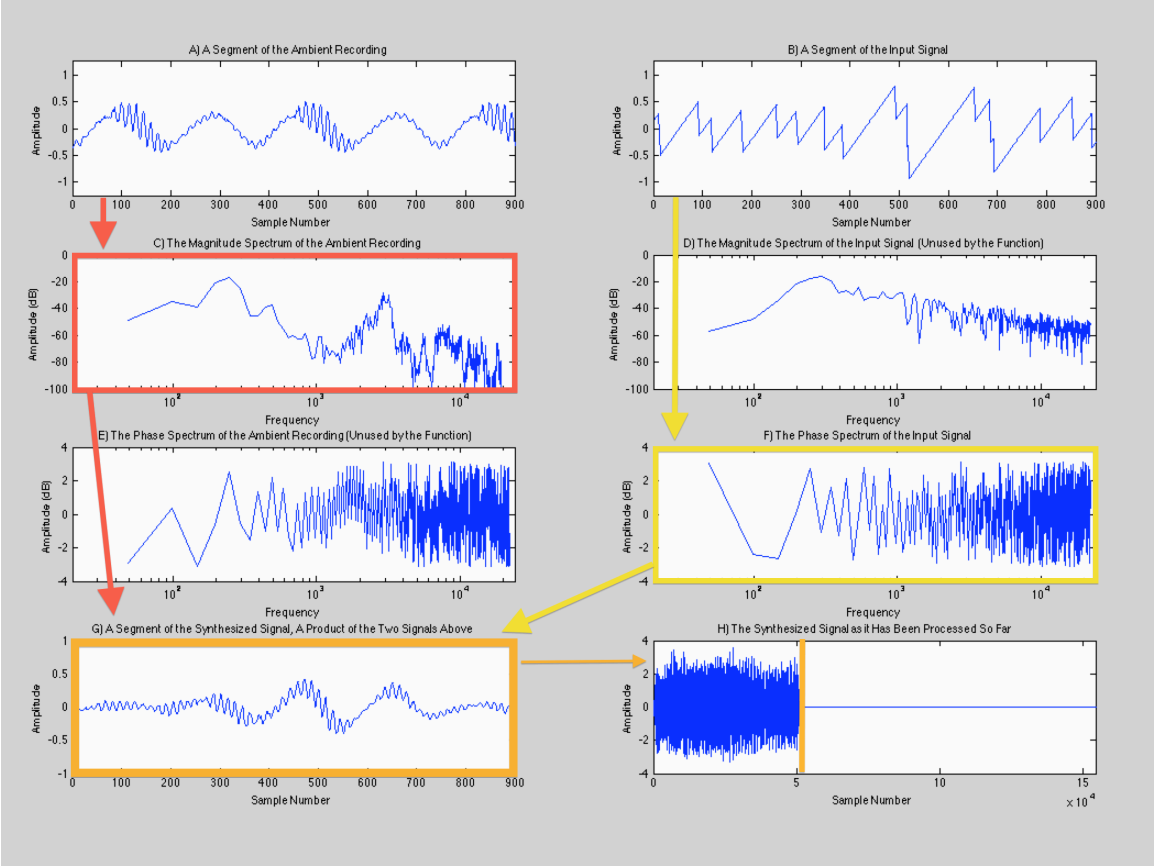
complex signal = magnitude * e^(j(phase))

where j is the square root of -1 and the number e is about 2.71828.

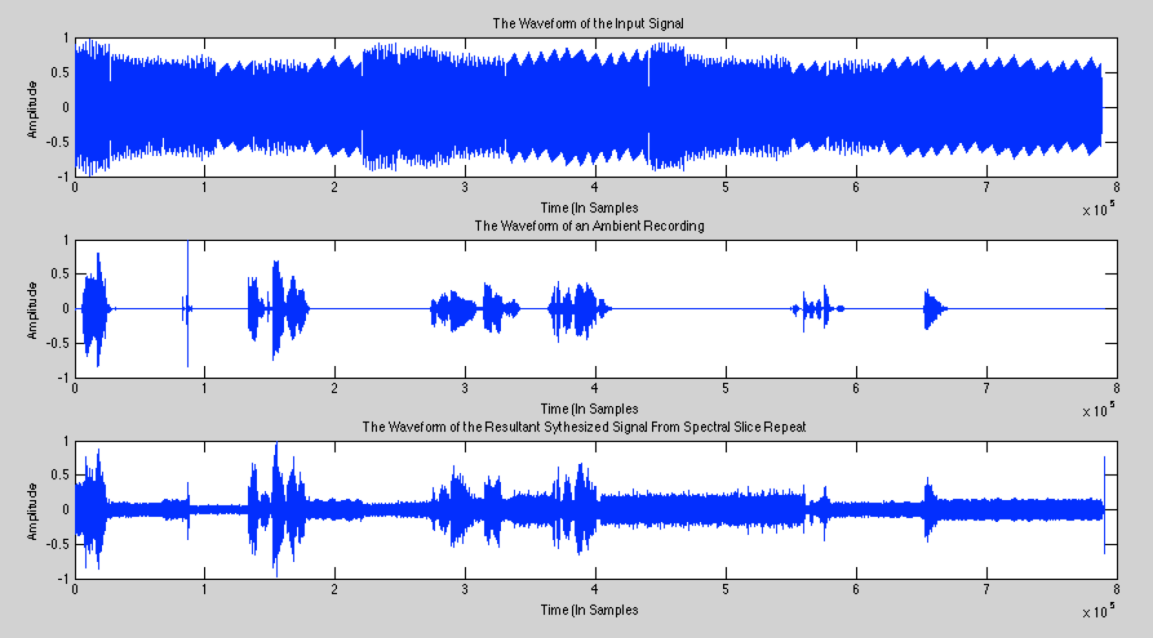
Matlab's ifft function is then used to transform this new segment into the time domain, using an equation identical to the one implemented by the fft function except there is no negative sign in front of the j.

The signal flow of these segments is illustrated in the figure below. The ambient_recording_segment is in red, the input_signal_segment is in yellow and the synthesized_segment is in orange. Notice that the synthesized_segment's major peaks and troughs follow those of the sawtooth wave in the input signal, but the finer contours of the wave match that of the ambient recording. If you have spent any time looking at audio waveforms, it should be plain that this shape indicates the signal's fundamental frequency matches the input_signal and has a timbre quite similar to the ambient_recording.

Each synthesized segment is added to the previous segment, overlapping it by 90%. This process continues until the total length of the ambient_recording and input_signal are analyzed. When the full synthesized_signal is produced it is sent back to the World_Around function.



As mentioned before, the only way spectral_splice, and spectral_splice_repeat differ is in the way they handle quiet sections of the ambient recording. Each time spectral_splice_repeat grabs segment of the ambient_recording it calculates its rms value (in the frequency domain). If the segment's rms value exceeds a set threshold, a copy of the segment is saved as the variable last_fft_magnitude_segment, and then used to contribute to the synthesized signal. Each time the segment is saved it replaces the previous segment saved to the last_fft_magnitude_segment variable. If a segment's rms value does not exceed the threshold, the last_fft_magnitude_segment variable is used to create the synthesized signal The function keeps using last_fft_magnitude_segment until it is replaced by the next segment that exceeds the threshold. The figure below plots the waveform of a synthesized_signal produced by spectral_splice_repeat as well as an input_signal and ambient_recording. Two things are clear from the diagram: the input_signal's magnitude does not influence the magnitude of the synthesized signal, and the synthesized signal continues to sound during the corresponding quiet sections of the ambient recording.



Examples

Examples of signals have been provided to demonstrate the functions performance at different quiet_compensation settings and in different environments. The same input_signal is used in each case. Compairing the ambient_recording and the output_signal is the most interesting, however the compressed signal and synthysizd_signal are also included in order to demonstrate signal flow.

In examples 1-3 quiet_compensation is set to 1.

In example 4 quiet_compensation is set to 2.

In example 5 and 6 quiet_compensation is set to 3.

In example 7-9 quiet_compensation is set to 1 again. Just to show that it is the best setting.

References

David Byrne, 2012. How Music Works. Edition. McSweeney's.

www.gsmarena.com. 2011. www.gsmarena.com. [ONLINE] Available at: http://www.gsmarena.com/mobile_phone_usage_survey-review-592p7.php. [Accessed 22 May 13].

www.mq.edu.au. 2008. Analog and Digital Sound. [ONLINE] Available at: http://clas.mq.edu.au/acoustics/frequency/analog_digital.html [Accessed 22 May 13].

www.utexas.edu. 2010. Representation of Waves via Complex Numbers. [ONLINE] Available at: http://farside.ph.utexas.edu/teaching/315/Waves/node72.html. [Accessed 22 May 13].