Audio Transport

Trevor Henderson

May 19, 2017

Abstract

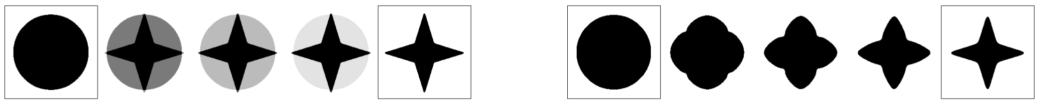
This paper describes a new technique to interpolate between audio signals. This technique uses optimal transport to move the pitches in one signal to the pitches in another. Audibly this sounds like a generalized portamento, or glide between sounds. The main contributions of this paper are methods to remove the different types of phasing distortion that can occur in a naive transport. A real time implementation of the algorithm produces musical results.

Code: https://github.com/sportdeath/Vocoder.
Video: https://youtu.be/PQGV0fk3Gww

1 Introduction

A portamento is a musical term used to describe one pitch sliding into another. The portamento can be found in almost all genres of music, whether it is a slur in a vocal passage or in the twang of a slide guitar. It provides smooth way to transition between notes and even chords.

This effect, though widely used, is limited in its use by the physical design of the instrument. Most acoustic keyboard instruments for example, like the piano or organ, can't glide at all because notes are formed from a discrete set of strings or pipes. Digital keyboards can produce a glide with a pitch wheel. The pitch wheel can change the playback rate of a sample as it plays or use spectral time stretching to change the pitch of a prerecorded audio sample. However, this system does not make it easy to slide between chords of different shapes. And moreover, it is impossible to slide between sounds with different timbres.



(a) Linear Interpolation (b) Optimal Transport Interpolation

Figure 1: From *Convolutional Wasserstien Distances: Efficient Optimal Transportation on Geometric Domains* by Justin Solomon et al, 2015 [11]

This paper describes a method to produce a portamento not only between sounds with different timbres but between sounds from different instruments entirely. It does this using optimal transport, a geometric technique typically found in applications to computer graphics and machine learning.

The optimal transport problem seeks a map between two probability masses in $\mathbb{R}^n$ that minimizes the work done on the mass. Mass in the transport must physically move through space and all mass must be preserved. I find that optimal transport is best explained visually as in figure 1 from Justin Solomon et al [11]. Efficient solutions to compute an optimal transport mapping is an open problem for $n > 1$. However, optimal transport on the real line has a closed form solution.

We will use optimal transport on the real line to find a mapping between the spectra of two audio sources that minimizes the distances that the pitches move. From this assumption we can guarantee that no pitches will cross each other.

2 Contributions

I developed a new technique to interpolate between two audio signals using optimal transport. The optimal transport problem is usually only defined only for positive real inputs. The spectra of an audio signal however is complex. It is natural that the transport be done on the absolute values of these complex numbers, but this begs the question of what is to be done with the phase.

This paper describes several types of distortion that can occur because of phasing issues and how to resolve them. These problems are similar to those experienced in phase vocoders — the technology that allows an audio signal's speed to be changed without changing the pitch. However the solutions posed in this paper improve upon the techniques used in phase vocoders and take advantage of the specifics of this problem set up. [8]

These are the main techniques used in the algorithm which are new to this paper:

1. A method to segment and audio spectrum into groups whose local phase relations have audible significance.

2. A method to update the phase of a pitch shifted signal given a source and destination.

Additionally we also provide a real-time implementation of the algorithm for audible demonstration.

3 Related Work

Many audio tools exist that modulate the pitch of an audio signal. The phase vocoder, also given other names in application like 'time-warping,' became practical in the early 2000's, is now pervasive in all types of music production [8] [1]. It is useful to piece together samples played in different keys, or to fine tune a recording. The phase vocoder is also the technology that enables auto-tune — an effect which swept through the music industry in the mid-2000's and is still popular today.

Several audio tools work to manipulate the pitches within an audio signal separately. Melodyne is a popular tool in the music industry for correcting the pitches of notes. In 2008, Melodyne released a feature to adjust the pitches of individual notes in polyphonic music, allowing a user to rewrite the harmony of a song [10]. This technology works on a static audio file, but in 2012 a product named Zynaptiq allows for different bands of an audio stream to be pitch shifted [12].

However neither of these products pitch shift on with granularity that is needed to slide between notes of different timbres. In addition, they rely on the user to select which notes they would like to tune. While this is an extremely useful feature in the studio, it is hard to control in the live setting.

Optimal transport has been used for audio processing before. However so far as I can tell, no paper has synthesized sound with optimal transport. Most of the existing papers in this cross section use optimal transport to help transcribe music. Using a distance metric based on the harmonic series seems to be useful for determining fundamental of a note [5] [3].

4 Algorithm Overview

4.1 STFT

Like almost all spectral algorithms, this one begins with a short-time Fourier transform (STFT) and ends with it's inverse (ISTFT). At the cost of reduced temporal resolution, the STFT gives us access to the spectral components of the signal in time. The following summarizes the relavant properties of the STFT for this problem as described by Julius O. Smith III in *Spectral Audio Signal Processing* [7].

The STFT performs a discrete time Fourier transforms on windows of size M of the original signal $x(n)$. These windows are separated by a hop size of R. The τth frame of the STFT is

$$X(\omega, \tau) = \mathrm{DTFT}(\tilde{x}(n, \tau)w(n)) \tag{1}$$

where

$$\tilde{x}(n, \tau) = x(n + \tau R) \tag{2}$$

is a buffer of the original signal, shifted R samples from the previous buffer and $w(n)$ is the synthesis window. $w(n)$ is defined to be nonzero for M samples, $n \in \left[-\frac{M-1}{2}, \frac{M-1}{2}\right]$, so only the M most recent samples of $x(n)$ need to be kept in memory.

After some computation, we produce frames $Y(\omega, \tau)$ which we want to use to synthesize the signal $y(n)$. We perform this ISTFT using the weighted overlap add method. As the name suggests, this method works by overlapping adjacent frames of the IDTFT, weighting them by an analysis window $f(n)$ and then adding them together. The weighting helps to suppress audible discontinuities at frame boundaries which prevents artifacts in highly nonlinear filters such as the one this paper describes.

$$\tilde{y}(n, \tau) = \mathrm{IDTFT}(Y(\omega, \tau)) \tag{3}$$

$$y(n) = \sum_{\tau} f(n - \tau R)\tilde{y}(n - \tau R, \tau) \tag{4}$$

Note that synthesizing a single sample of $y(n)$ requires overlapping $\frac{M}{R}$ frames which adds a latency of M samples between y and it's sources.

In the absence of spectral modifications, we can achieve perfect reconstruction of a signal transformed by the STFT and ISTFT given that the synthesis and analysis windows obey

$$\sum_{\tau} w(n - \tau R) f(n - \tau R) = 1, \ \forall n \in \mathbb{Z} \tag{5}$$

I chose to use the square root of the Hann window as both the analysis and frequency window which gives perfect reconstruction when $R = \frac{M}{2k}$ for $k \in \mathbb{Z}$.

$$w(n) = f(n) = \cos(\pi n / M) \tag{6}$$

For actual computation we replace, the DTFT and IDTFT with the FFT and IFFT respectively. The FFT has M bins $n \in \left[-\frac{M-1}{2}, \frac{M-1}{2}\right]$. The frequency of the nth bin is given by

$$\omega = 2\pi \frac{f_s n}{M} \tag{7}$$

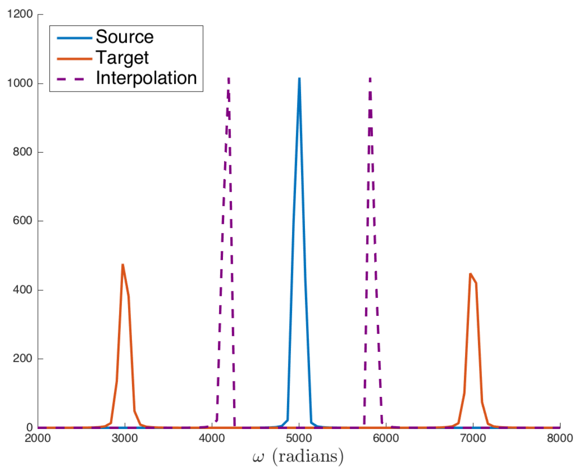
where f_s is sampling rate of the signal. Since the audio signal $x(n)$ is real, $X(\omega)$ is conjugate symmetric, so we only need to retain the positive bins $\left[0, \frac{M-1}{2}\right]$ of the FFT.

4.2 Spectrum Segmentation

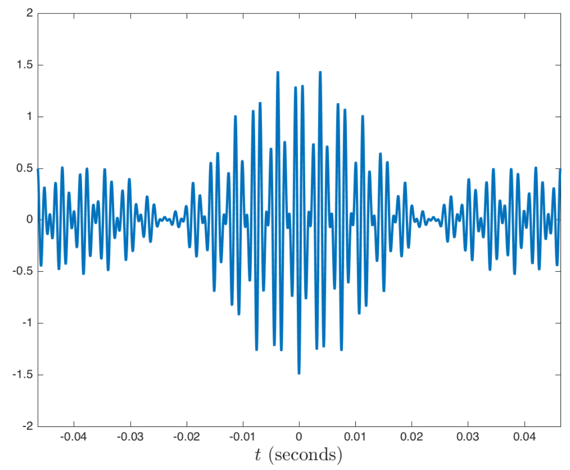
Once we have the spectrum X_0 from one window of the STFT, we then want to transport it to another spectrum X_1 from a separate STFT running in parallel. However performing a transport across all bins results in distortion as seen in figure 2. This distortion is known as vertical incoherence in phase vocoder algorithms. It occurs when local phase relations pertaining to a particular frequency component are lost [8].

A technique that is used to fix this problem in phase vocoders is to lock the phases of particular regions of the spectrum [8]. For this problem, this means we must do the transport on chunks of the spectrum rather than the individual bins. All that is needed now is an algorithm to segment the spectrum into pieces whose phase relations we want to maintain.

Phase vocoder algorithms often use peak finding to determine these regions [8]. However, peak finding is not very robust to noise. Any small oscillation in the transform can cause an unnecessary split and produce vertical incoherence. This does not seem to effect the performance of the phase



(a) Interpolated Spectra



(b) Interpolated Waveform

Figure 2: If we perform a transport across all bins, sinusoids smeared across multiple bins can be split in two (2a). The resulting waveform (2b) should the sum of two sinusoids, but it experiences distortion known as *vertical incoherence.*

vocoder because all of these regions are moving to the same pitch. But the effect becomes much more noticeable when suddenly all regions are moving to entirely different pitches. Adding a noise threshold not only makes the segmentation less elegant, it also begins to ignore high frequency components, which typically have a much smaller amplitude.

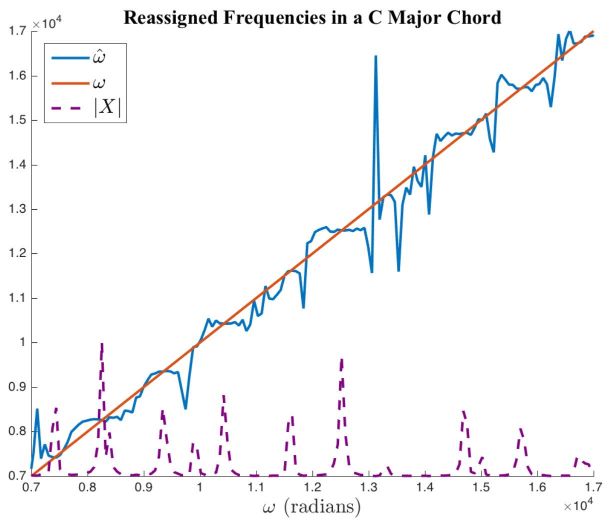


Figure 3: The reassigned frequencies in a C Major chord played on the piano

Instead, I determine where to split the window based on the *reassigned frequency*, $\hat{\omega}(\omega)$. The reassigned frequency as shown in figure 3 is a more accurate notion of the frequency a bin is actually oscillating at. A sinusoid smeared over multiple bins for example, should have a constant reassigned frequency, because the smeared amplitudes all support the center frequency. Likewise, many of the peaks in the spectrum of the piano chord in figure 3 align with relatively flat sections of the reassigned frequency [4].

Once the reassigned frequency has been computed, we can use algorithm 1 to segment the spectrum into a set S of spectral pieces $s(i)$. Each piece s is defined by its center of mass $c(s)$, its set of bins $U(s)$, and a mass $\rho(s)$. The center of the mass is picked so that it is the bin whose frequency is closest to the reassigned frequency. The mass of each segment is a normalized sum of its components

$$\rho(s) = \frac{\sum_{n \in U(s)} |X(n)|}{\sum_n |X(n)|} \tag{8}$$

Algorithm 1 Segment Spectrum

```
$S \leftarrow \{\}$
$s \leftarrow \emptyset$
**for** each bin $n > 0$ **do**
    $U(s) \leftarrow U(s) \cup n$

    **if** $\hat{\omega}(n-1) - \omega(n-1) > 0$ and $\hat{\omega}(n) - \omega(n) < 0$ **then**
        // $\hat{\omega}$ crosses down through $\omega$
        // Make a center of mass
        $c(s) \leftarrow n - 1$

    **else if** $\hat{\omega}(n-1) - \omega(n-1) < 0$ and $\hat{\omega}(n) - \omega(n) > 0$ **then**
        // $\hat{\omega}$ crosses up through $\omega$
        // Split the mass
        $S \leftarrow S \cup \{s\}$
        $s \leftarrow \emptyset$
    **end if**
**end for**
```

This normalization occurs so that the sum of masses equals 1 for optimal transport:

$$\sum_{s \in S} \rho(s) = 1 \tag{9}$$

Figure 4 shows a segmentation performed by the algorithm. The figure shows that many of the center of mass marked in green align exactly with the peak. However the small peak near $\omega = 8000$ radians $\cdot\, \mathrm{s}^{-1}$ is ignored because it is dominated by the larger peak next to it which still exerts influence.

A slight modification must be made to this algorithm to account for audio where the relation between frequency and reassigned frequency is relatively linear as shown in figure 5. In these cases, slight perturbations in the reassigned frequency with cause it to fluctuate over and under the bin frequency producing unnecessary segmentations. We resolve this by adding a boundary region whose width is equal to the width of an FFT bin, $2\pi\frac{f_s}{M}$. The algorithm is essentially the same only we form masses when we cross up through the entire boundary region and define our center of masses when we cross down through the entire boundary region.

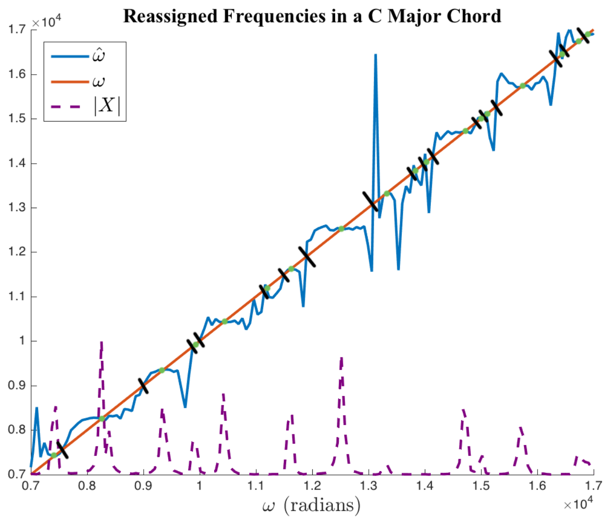


Figure 4: The segmentation produced by algorithm 1. Centers of mass are marked in green and the boundaries between masses are marked in black.

The reassigned frequency can be computed as follows [4]:

$$\hat{\omega}(\omega) = \omega + \Im\left\{\frac{X_{\mathcal{D}}(\omega) \cdot X^*(\omega)}{|X(\omega)|^2}\right\} \tag{10}$$

where $X_{\mathcal{D}}$ is the spectrum of the input signal $x(n)$ windowed with $w_{\mathcal{D}}(n) = \frac{d}{dt}w(n)$:

$$X_{\mathcal{D}}(\omega) = \text{FFT}\left(w_{\mathcal{D}}(n)x(n)\right) \tag{11}$$

The derivative of our chosen window, the square root of the Hann window is:

$$w_{\mathcal{D}}(n) = -\frac{\pi f_s}{M}\sin\left(\frac{\pi n}{M}\right) \tag{12}$$

4.3 Transport

After two audio signals have been segmented into S_0 and S_1 we then compute a transformation matrix $T \in \mathbb{R}^{|S_0|\times|S_1|}$ whose entries represents the transfer of mass from one segment to the other. We want to construct this matrix so

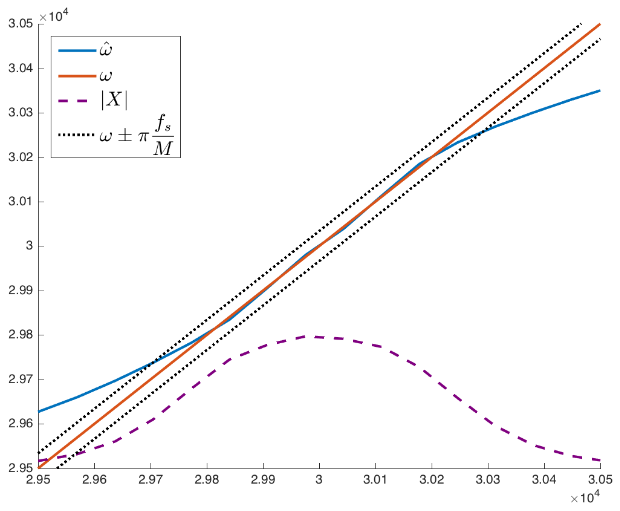


Figure 5: The reassigned frequency of a linear chirp — a sinusoid with linearly increasing amplitude. The reassigned frequency is approximately linear over the chirps range. The dotted black lines show a region where small fluctuations do not produce new segmentations.

that the total movement of mass (or work) is minimized. We do not want to transport a negative mass or more mass than is available:

$$0 < T(i,j) \leq \max(\rho(s_0(i)), \rho(s_1(j))) \tag{13}$$

Additionally, we want to constrain that all of the mass receives an assignment:

$$\rho(s_0(i)) = \sum_j T(i,j)\ \forall i \tag{14}$$

$$\rho(s_1(j)) = \sum_i T(i,j)\ \forall j \tag{15}$$

Given these constraints we want to find the assignment T that minimizes the 2-Wasserstein distance between the segments [11].

$$\min_T \sum_{i,j} T(i,j)|c(s_0(i)) - c(s_1(j))|^2 \tag{16}$$

Algorithm 2 Compute Transformation Matrix

```
$T \leftarrow 0$
$i, j \leftarrow 0$
$\rho_0 \leftarrow s_0(i)$
$\rho_1 \leftarrow s_1(j)$
**while** $i < |S_0|$ and $j < |S_1|$ **do**
    **if** $\rho_0 < \rho_1$ **then**
        $T(i, j) \leftarrow \rho_0$
        $i \leftarrow i + 1$
        $\rho_1 \leftarrow \rho_1 - \rho_0$
        $\rho_0 \leftarrow \rho(s_0(i))$
    **else**
        $T(i, j) \leftarrow \rho_1$
        $j \leftarrow j + 1$
        $\rho_0 \leftarrow \rho_0 - \rho_1$
        $\rho_1 \leftarrow \rho(s_1(j))$
    **end if**
**end while**
```

We compute such a T with algorithm 2. This algorithm takes advantage of the fact that in a minimized mapping, no mass passes over any other mass. The leftmost unmapped mass in one segmentation must always come from the leftmost unmapped mass in the other. So we can greedily assign masses from left to right.

This algorithm runs linearly and guarantees that T will be sparse with no more than $2N - 1$ entries. This algorithm is visualized in figure 6

4.4 Phase Accumulation

After we have a transport map, we need to actually repitch the spectral segments. When we move a segment of the spectrum to a new pitch, the phases within that section rotate either slower or faster. This change does not make an audible difference within a window itself, but it can cause interference in the overlap between windows known as *horizontal incoherence* seen in figure 7.

One solution to this in the realm of phase vocoders is to estimate the phase of the next frame based on its instantaneous frequency [8]. If a sinusoid with

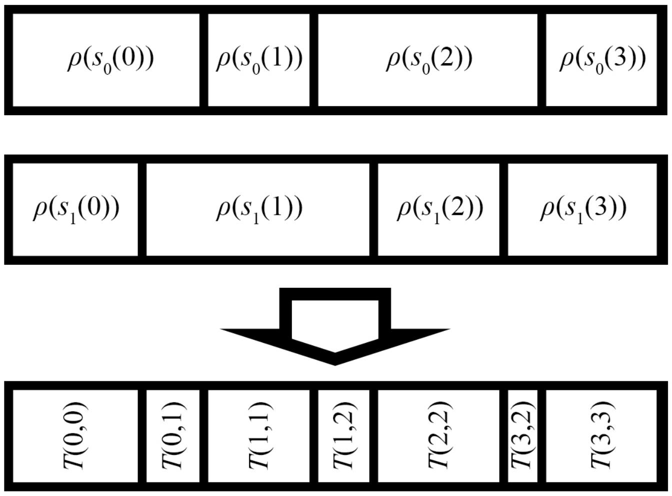


Figure 6: A visualization of the 1D optimal transport algorithm. The width of the bins represents the amount of mass in that bin.

phase $\angle X(\omega, \tau)$ is oscillating with instantaneous frequency $\hat{\omega}(\omega)$ over frame τ, we would expect the phase of the next frame to be

$$\angle X(\omega, \tau + 1) = \angle X(\omega, \tau) + \frac{R}{f_s}\hat{\omega}(\omega) \tag{17}$$

This model is good for signals which do not undergo rapid changes in frequency. However it does a poor job of dealing with transients in audio that can happen from sharp note attacks and percussive sounds. To cope with this, some phase vocoders employ phase reinitialization steps in frames where transients are detected. Even with this step additional phasing can still occur [8].

Fortunately however, we have at our disposal the phases of not one but two pitches. If we are interpolating along the transport map with interpolation factor k, then for a particular assignment between s_0 and s_1 we know the center frequency of the mass will oscillate at frequency:

$$\hat{\omega}' = (1 - k)\hat{\omega}(c(s_0)) + \hat{\omega}(c(s_1)) \tag{18}$$

The accumulated phases $\Theta_0(\omega, \tau)$ and $\Theta_1(\omega, \tau)$ each change proportional to ω_0 and ω_1. So to get a new phase Θ' whose rate of change is proportional to

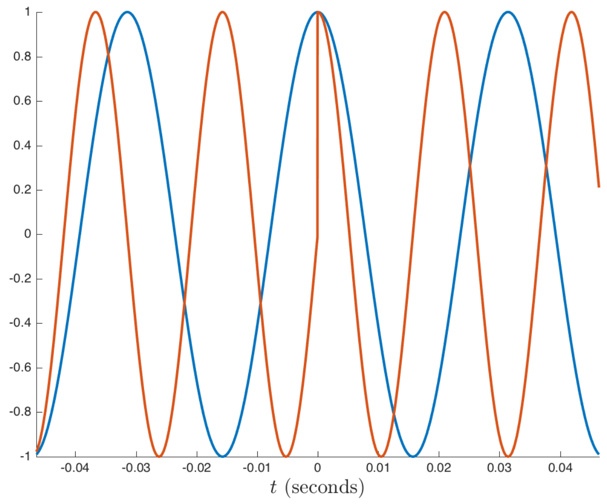


Figure 7: The pitch of the original signal in blue is increased to be that of the orange signal. The window boundary at $t = 0$ experiences horizontal incoherence.

$\hat{\omega}'$, we can linearly interpolate between Θ_0 and Θ_1.

$$\Theta' = (1 - k)\Theta_0(\omega, \tau) + k\Theta_1(\omega, \tau) \tag{19}$$

We calculate the accumulated phase so that it increases with respect to $\hat{\omega}$, but round it to maintain the exact phase. This rounding is essentially a reinitialization every step, which we get away with because of the two sources.

$$\Theta(\omega, \tau) \approx \sum_{t=0}^{\tau} \frac{R}{f_s} \hat{\omega}(\omega, \tau) \tag{20}$$

$$\Theta(\omega, \tau) = \Theta(\omega, \tau) \pmod{2\pi} \tag{21}$$

4.5 Resynthesis

After we have segmented, transported, and accumulated the phases, we can finally synthesize an output fourier transform $Y(n)$ as in algorithm 3. For every assignment, we find the new center frequency, and new center phase.

Algorithm 3 Resynthesize

```
for all $n$ do
    $Y(n) \leftarrow 0$                                              ▷ Clear the output
end for

for $i$, $j$ do

    $c' \leftarrow (1-k)c(s_0(i)) + kc(s_1(j))$                       ▷ $c'$ is the new center index
    $\Theta' \leftarrow (1-k)\Theta(c(s_0(i))) + k\Theta(c(s_1(j)))$    ▷ $\Theta'$ is the new center phase

    for $n \in U(s_0(i))$ do

        $n' \leftarrow n + \hat{n} - c(s_0(i))$                          ▷ $n'$ is the new bin index
        $\theta' \leftarrow \Theta_0(n) + \Theta' - \Theta_0(c(s_0(i)))$    ▷ $\Theta'$ is the new bin phase

        $Y(n') \leftarrow Y(n') + (1-k)\frac{T(i,j)}{\rho(s_0(i))}|X_0(n)|e^{i\theta'}$    ▷ Linearly interpolate

    end for
    for $n \in U(s_1(j))$ do

        $n' \leftarrow n + \hat{n} - c(s_1(j))$                          ▷ $n'$ is the new bin index
        $\theta' \leftarrow \Theta_1(n) + \Theta' - \Theta_1(c(s_1(j)))$    ▷ $\Theta'$ is the new bin phase

        $Y(n') \leftarrow Y(n') + j\frac{T(i,j)}{\rho(s_1(j))}|X_1(n)|e^{i\theta'}$    ▷ Linearly interpolate

    end for
end for
```

We then adjust all the phases within the segment to maintain local phase relations. Finally we linearly interpolate between all of the segments according to the interpolation factor.

5 Implementation

I implemented the above algorithm in real-time as an audio effect in C++. The program listens to two streams of audio which can be sent to it from a any digital audio workstation. It then produces a new stream of audio whose spectrum is transported between the two according to a MIDI value. The program depends on the library FFTW (Fastest Fourier Transform in the West) [6] for Fourier transforms, and the open source libraries PortAudio [2] and PortMidi [9] for audio and MIDI handling respectively.

I work with a window size of 4096 samples and a sampling rate of 44,100 hertz. This produces a delay of 10 milliseconds which is audible. However the delay only exists on from the perspective of the source audio. There is no perceivable delay from the perspective of the midi controller because the MIDI value is read right before a frame is synthesized.

Code: https://github.com/sportdeath/Vocoder.

Video: https://youtu.be/PQGV0fk3Gww

6 Conclusion

As seen in the implementation video, the algorithm produces undistorted signals for a variety of inputs: monophonic - polyphonic, acoustic - synthesized, bright - dark.

One of the most noticeable artifacts that still exists is the interference that results from a single pitch being mapped to many places. As the note splits and moves to new locations it interferes with itself for the brief period of time where the splits overlap. This artifact causes a drop in volume at the very start or end of a transport. This type of interference has plagued music far before computers. If adjacent semi-tones on a piano are played together you can hear beating. As of now, I have no idea how to fix that other than performing a transport that only maps a frequency to a single other frequency. But that takes some of the fun away. And in the end the distortion doesn't sound harsh and digital and in many cases the volume dip is barely noticeable.

Overall the effect produced a variety of interesting sounds. This effect has potential both in the studio and live setting. It can be used by DJs to transition between songs or could be used by guitarists to swap between dualing solos. It functions as a new way to filter between sounds of different timbres. Moreover some of the sounds it produces sound simply unexplainable and I'm excited to explore their possibilities.

References

[1] Ableton. Audio clips, tempo, and warping. `https://www.ableton.com/en/manual/audio-clips-tempo-and-warping/`, 2017.

[2] Phil Burk. Portaudio. `http://www.portaudio.com`, 2017.

[3] Filip Elvander, Stefan Ingi Adabjornsson, Johan Karlsson, and Andreas Jakobsson. Using optimal transport for esimating inharmonic pitch signals, 2016.

[4] Kelly R. Fitz and Sean A. Fulop. A unified theory of time-frequency reassignment. *Digital Signal Processing*, 2005.

[5] Remi Flamary, Cedric Fevotte, Nicolas Courty, and Valentin Emiya. Optimal spectral transportation with application to music transcription. In *29th Conference on Neural Information Processing Systems (NIPS 2016)*, 2016.

[6] Matteo Frigo. A fast fourier transform compiler. In *Proceedings of the 1999 ACM SIGPLAN Conference on Programming Language Design and Implimentation (PLDI '99)*, 1999.

[7] Julius O. Smith III. *SPECTRAL AUDIO SIGNAL PROCESSING*. Center for Computer Research in Music and Acoustics (CCRMA), 2011.

[8] Jean Laroche and Mark Dolson. Improved phase vocoder time-scale modification of audio. *IEEE Transactions on Speech and Audio Processing*, 7(3), 1999.

[9] PortMidi. Platform independent library for midi i/o. `http://portmedia.sourceforge.net/portmidi/`.

[10] Mike Senior. Celemony melodyne dna editor. http://www.soundonsound.com/reviews/celemony-melodyne-dna-editor, 2009.

[11] Justin Solomon, Fernando de Goes, Gabriel Peyre, Marco Cuturi, Adrean Butscher, Andy Nguyen, Tau Du, and Leonidas Guibas. Convolutional wasserstein distances: Efficient optimal transportation on geomtric domains. In *SIGGRAPH 2015*, 2015.

[12] Zynaptiq. Pitchmap. http://www.zynaptiq.com/pitchmap/, 2017.