Hammond Organ Effects Synthesis

Evan Savage

MUMT 618
Professor: Dr. Gary Scavone

December 11, 2018

1 Introduction

The Hammond Organ, first manufactured in 1935, is a widely-celebrated instrument in many musical genres. It was first introduced as a substitute to expensive pipe organs, allowing for a rich output of sound in a cheaper and portable package. The Hammond was dominantly present in church settings until the 1950s and 60s, when exploratory artists in the realms of jazz, blues, and rock were able to find a place for the organ's sound in their various compositions. Notable performers such as Jimmy Smith and Keith Emerson pushed the boundaries of the instrument, motivating numerous others to follow suit.

Tonewheels or ferromagnetic metal disks, rotate near electromagnetic pickups to produce the sound of the organ. Smooth ridges around the rim pass close to and then past the individual pickups, inducing opposing currents that pass through a coil producing a fundamental frequency. Varying the pattern and number of edges on each tonewheel correspond to different frequencies. There are 91 separate tonewheels, tuned at the lowest to C1 (32.7 Hz) up to F7 (5919.9 Hz). The first octave is composed of richer tones featuring strong 3rd and 5th harmonics, closer to square wave responses while the rest are sinusoidal. The Hammond acts as an additive synthesizer, driving multiple tonewheels in parallel as the player compresses multiple notes and changes the available controls. The tonewheels lie very close to one another in physical space, which has led many to believe that crosstalk between the respective pickups contributes to the Hammond's characteristic sound.

The main feature of the Hammond is a set of 9 drawbars that control the amplitudes of pseudo-harmonic frequencies relative to all 61 keys on the keyboard. Tuned to the 12-tone equal temperament scale, each of the nine drawbars corresponded to semi-tonal offsets of -12, 7, 0, 12, 19, 24, 28, 31, 36 respectively. Each individually adjustable drawbar has nine positions that account for the gain of the respective harmonics in increments of -3 dB per step (0, -3, -6, -9, -12, -15, -18, -21, -inf). When a drawbar is pushed all the way in, the respective harmonic will not be heard (-inf dB adjustment), opposed to its notable presence when it is fully extended (0 dB adjustment).

For this project, I analyzed the implementation proposed by Dr. Kurt James Werner and Dr. Jonathan S. Abel in their Applied Sciences paper from 2016, "Modal Processor Effect Inspired by Hammond Tonewheel Organs." Adapting their modal reverberator

processing chain suggested in previous papers, they synthesized the controls and perceptual characteristics of the Hammond as an effects processor. Some effects such as vibrato and tonewheel distortion were presented, though I was unable to successfully implement them. Using MATLAB, I have implemented the drawbar controls and crosstalk between tonewheel pickups that contribute to the Hammond's unique sound.

2 Background

2.1 Modal Processor Overview

The architecture of a modal reverberator models a collective impulse response as the summation of parallel vibrational mode responses. The original composition was suggested for room mode responses, though the architecture can be adapted for the Hammond effects processor. Denoting the impulse response as $h(t)$, is can be expressed as a combination of mode responses.

$$h(t) \;=\; \sum_{m=1}^{M} h_m(t) \tag{1}$$

The system is summed across M total modes, where each individual mode impulse response, $h_m(t)$, is part of a convolution with an input $x(t)$.

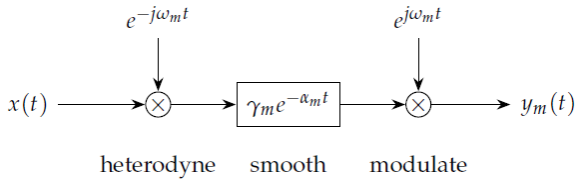
$$y_m(t) \;=\; h_m(t) \;*\; x(t) \tag{2}$$

The final output, $y(t)$, is formed as a parallel combination of each modal convolution.

$$y(t) \;=\; \sum_{m\;=\;1}^{M} y_m(t) \tag{3}$$

Through a cascade of heterodyning, smoothing (convolution), and demodulating operations, audio input $x(t)$ can be processed across any number of mode responses.

Figure 1. Individual mode response operations



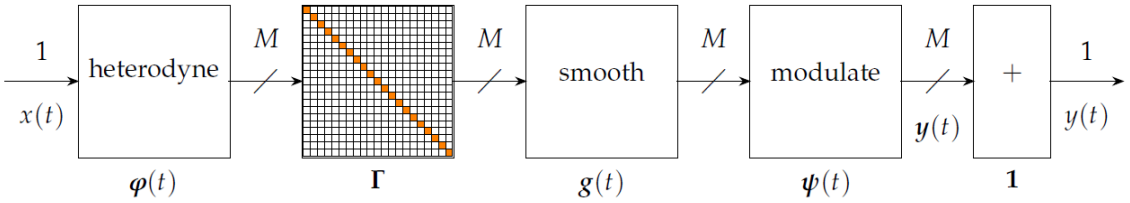
Source: "Modal Processor Effects Inspired by Hammond Tonewheel Organs"

Heterodyning in the context of this implementation refers to an amplitude modulation. The input signal with fundamental frequency f_c, when multiplied with the complex exponential, $e^{-j\omega_m t}$, will be offset in frequency by $f_c \pm f_m$, where $\omega_m = 2\pi f_m$. The smoothing

step involves another multiplication with an amplitude, γ_m, and exponential seen in $\gamma_m e^{-\alpha_m t}$. The amplitude and damping constant, α_m, correspond to the damping properties of the object being synthesized. Finally, each scaled mode response is demodulated to the output via another multiplication with the complex exponential, $e^{j\omega_m t}$.

In order to view the overall structure of the modal processor, it can be viewed as a series of matrix multiplications:

Figure 2. Modal reverberator architecture



Source: "Modal Processor Effects Inspired by Hammond Tonewheel Organs"

The heterodyning and modulating (demodulating) sinusoids described above are stacked into columns of height M as $\varphi(t)$ and $\psi(t)$ respectively. The smoothing step is handled by two separate matrices, Γ and $g(t)$. Γ is a diagonal $[M \times M]$ matrix with amplitudes corresponding to each individual mode gain, while $g(t)$ is another column of height M consisting of the complex damping operations. As shown in equation (3), the final output $y(t)$ is formed as a summation of each mode response $y_m(t)$ and represented by the final matrix in Figure 2.

2.2 Hammond Effects Processor Overview

The Hammond organ operates as an additive synthesizer, driving a subset of its 91 tonewheels in parallel depending on which of its 61 keys are compressed and how the drawbars are positioned. Thus, each key and tonewheel can be represented in another series of matrix multiplications (Figure 3) similarly oriented to Figure 2.

The heterodyning step is essentially equivalent, where the single input signal $x(t)$ is now modulated across the frequencies corresponding to the 61 keys on the Hammond.

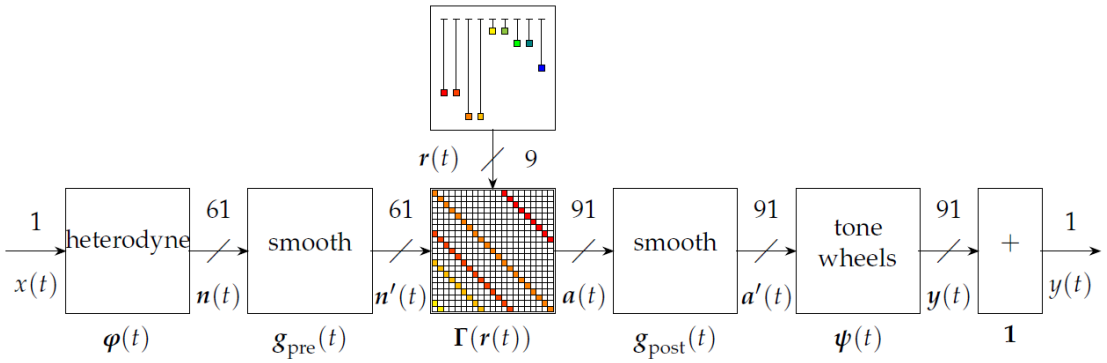
$$n(t) \ = \varphi(t)x(t) \tag{4}$$

The column of key frequencies is smoothed prior to the gain matrix by a set of element-wise convolutions with $g_{pre}(t)$. The convolutions can also be represented as normal time domain filters to maintain the single input, $x(t)$, to single output, $y(t)$, chain of matrix operations.

$$n'(t) = g_{pre}(t) * n(t) \tag{5}$$

The drawbar controls are found in $\Gamma(r(t))$, representing the heart of the Hammond's functionality. Each color coded drawbar in $r(t)$ (Figure 3) is mapped to its corresponding diagonal of gain values in $\Gamma(r(t))$. Changing the position of each respective drawbar will

Figure 3. Hammond effects synthesis architecture



Source: "Modal Processor Effects Inspired by Hammond Tonewheel Organs"

in turn modify the amplitudes for the relative pseudo-harmonic offsets of the fundamental frequency in $x(t)$. Performing a matrix multiply with $n(t)$ results in the set of amplitudes passed to the tonewheels.

$$a(t) = \Gamma(r(t))n'(t) \tag{6}$$

The matrix mutiplication interpolates the frequencies of the 61 keys to the frequencies of the 91 tonewheels. Another element-wise smoothing operation takes place to set a relative unity gain for each tonewheel that will be driven. Again, time domain filters can be used here in place of the convolution.

$$a'(t) = g_{post}(t) * a(t) \tag{7}$$

Each mode output, $\boldsymbol{y}(t)$ is computed via the tonewheel (demodulation) step, which is an element-wise multiplication with the respective tonewheel frequencies.

$$\boldsymbol{y}(t) = a'(t) \odot \psi(t) \tag{8}$$

Finally, the modal responses are summed together to form the single output signal $y(t)$, representative of the additive synthesis of the Hammond organ.

$$y(t) = 1^\top \boldsymbol{y}(t) \tag{9}$$

3 Hammond Effects Implementation

Each of the equations listed in 2.2 were implemented in MATLAB via a series of element-wise and matrix multiplications, and successive digital filtering during the smoothing steps. Due to difficulties in the suggested processing from Werner and Abel, I was not able to implement some of the features found in their paper, such as the vibrato and memoryless pickup nonlinearities. Thankfully, I was able to successfully synthesize the modal frequencies with

basis tonewheel distortion, drawbar controls, and tonewheel crosstalk in the context of the processing architecture.

Through various adaptations, I was able to successfully process sinusoidal inputs leading to "hammondized" outputs. My code is able to be scaled to any number of desired modal responses, M, though it can quickly become very computationally demanding since each matrix operation happens across two-dimensional arrays that are mostly of size $[M \times t]$, where t is representative of successive samples from time 0 up until a specified duration in seconds, sampled at the audio rate, $f_s = 44.1$ kHz.

3.1 Modal Frequencies

When modulating over the organ's range, each mode acts as a bandpass filter, narrow enough to significantly reduce the frequencies of $x(t)$ that might lie between the 12-tone equal tempered frequencies that the organ keys are tuned to. Werner and Abel suggest expanding the amount of modes per semitone, represented as S=14, exponentially across the frequency range of the organ.

The lowest tonewheel frequency is 32.7 Hz, and the highest is 5919.9 Hz. Bounded within that range for the sake of authenticity, I set the first modal frequency f_1=40 Hz, also at the authors' suggestion. Seven octaves above f_1 is a frequency of 5120 Hz. Tuned to the 12-tone equal tempered scale, with 14 modes per semitone (S) across seven octaves, this leads to 1176 exponentially-spaced modes bounded within the frequency range of the organ. This range is roughly equivalent to the sinusoids available on 84 of the 91 tonewheels. The tuning of each mode can be found by

$$f_w = f_1 2^{w/(12S)} \tag{10}$$

where $w \in [1...1177]$ and again, $S = 14$. Thus, every matrix utilized in the processing is of height 1177, leading to a computationally demanding implementation for any input duration that is considerably long.

Each of the heterodyning and modulating sinusoids

$$\varphi(t) = \Re(e^{-j\omega_w t}) = \cos(\omega_w t) \tag{11}$$

$$\psi(t) = \Re(e^{j\omega_w t}) = \cos(\omega_w t) \tag{12}$$

are composed with $\omega_w = 2\pi f_w$. I decided to only use the real part of the sinusoids since the implementation was entirely executed in the time domain, and avoiding complex numbers greatly reduced the time of computation.

The lowest 12 tonewheels produce waveforms that are more square than sinusoidal. The corresponding modes encompass the first 144 indices of $\psi(t)$ while the rest are sinusoidal.

$$\psi(t) \;=\; \begin{cases} \frac{4}{\pi}\cos(\omega_w t) \;+\; \frac{4}{3\pi}\cos(3\omega_w t) \;+\; \frac{4}{5\pi}\cos(5\omega_w t) & w \;\in [1 \ldots 168] \\ \\ \cos(\omega_w t) & w \;\in [169 \;\ldots 1177] \end{cases} \tag{13}$$

Demodulation on the first set of modes in (13) leads to a small amount of distortion for the lower frequency pseudo-harmonic offsets present in the output.

Using the originally-suggested complex sinusoids led to a very distorted, incorrect output. Only modulating with the real sinusoids led to a much cleaner, harmonically-accurate result.

3.2 Drawbar Routing Matrix

The drawbar routing matrix is a sparsely filled square matrix featuring nine diagonals pertaining to the pseudo-harmonic offset values shown below. The offsets are said to be pseudo-harmonic since the first two are not positive integer multiples of the fundamental.

$$\boldsymbol{o} = [o_1 \dots o_9] = [-12, 7, 0, 12, 19, 24, 28, 31, 36] \tag{14}$$

In the same order, the nine drawbars individually control the level of the semitonal offsets. Each drawbar has nine positions, r_d where $d \in [1 \dots 9]$, corresponding to multiples of -3 dB increments until the fully compressed position where there is no relative harmonic output.

Table 1. Amplitude adjustment based on drawbar position

r_d	8	7	6	5	4	3	2	1	0
amplitude (dB)	0	-3	-6	-9	-12	-15	-18	-21	$-\infty$

Based on the position of r_d, I converted the decibel values to their respective amplitude gains, R.

$$R_{rd} = 10^{3(r_d - 8)/20} \tag{15}$$

Then, each entry in $\Gamma(r(t))$ can be expressed as

$$\gamma_{w,k}\,(t) \;=\; \sum_{d\,=\,1}^{9} R_{rd}\delta(w \;-\; k \;-\; o_d S) \tag{16}$$

where w denotes the same tonewheel modal index as before, and k is a modal offset of w, $k \in w + So$. Hence the routing matrix is of dimensions $[1177 \times 1177]$, as offsets in k above the modal range are truncated. $\delta(x)$ represents the Kronecker delta function

$$\delta(x) \;=\; \begin{cases} 1 & ,\; x \;=\; 0 \\ 0 & ,\; x \;\neq\; 0 \end{cases} \tag{17}$$

No two drawbar gains sum with one another due to the inclusion of $\delta(x)$, which causes the matrix to be so sparsely populated. I wrote a very simple external "kron.m" file for $\delta(x)$ which was used in coordination with my main script.

3.3 Smoothing Filters

The first smoothing filter, $g_{pre}(t)$, is included to prepare the incoming signal for the drawbar routing matrix multiplication. Werner and Abel suggest representing it with a column of ones, which acts as a moving average or low pass filter across each of the modes.

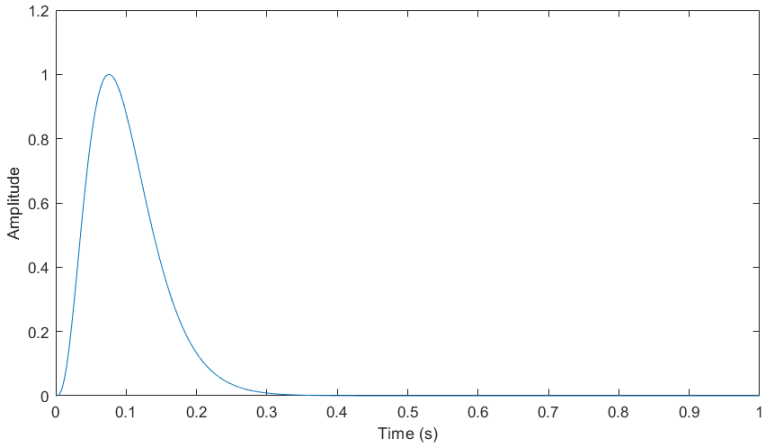
In earlier iterations, I was convolving each mode with $g_{pre}(t)$, which had to occur 1177 times and successive matrices had to be zero padded to account for the change in length of the output matrix due to the convolution. Thankfully, in an email exchange with Werner, he suggested to instead design digital filters in place of the convolution operations found in equations (5) and (7). The Z-transform of $g_{pre}(t)$ was simple to implement as an FIR filter, with the b and a coefficients set to "$(1/1177)$ones$(1, 1177)$" and "1" respectively.

The only information regarding $g_{post}(t)$ was that it should "create impulse responses with a linear ramp onset and a 200 ms decay — i.e., of the form $te^{-\alpha t}$" and that it should be applied twice. The z-transform of a time-multiplied exponential is

$$H(z) = \frac{Te^{-\alpha t}z^{-1}}{1 - 2e^{\alpha T}z^{-1} + e^{-2\alpha T}z^{-2}} \tag{18}$$

where T $(1/f_s)$ represents the discrete time sample. I used the "impz" function in MATLAB to continually augment the coefficients until an impulse filtered twice through the exponential resulted in the response in Figure 4.

Figure 4. $g_{post}(t)$ impulse response modeled in the form $te^{-\alpha t}$ twice-filtered



The impulse response's leading edge is close to linear ahead of the peak, and a 200 ms decay can also be seen. By implementing the convolution operations instead as digital filters, I was able to carry out all of the matrix operations without having to do any zero padding or concatenation, resulting in much shorter computation times.

3.4 Tonewheel Crosstalk

After smoothing the pseudo-harmonic amplitude outputs, $a(t)$, into $a'(t)$ (equation 7), another matrix multiply can be added to simulate crosstalk between adjacent tonewheel

pickups. Each of the mode filters act more as bandpass than a brick wall, so a minimal amount of crosstalk is already taking place.

$$a''_w(t) \;=\; Ca'_{w\ -\ S}(t) + a'_w(t) + Ca'_{w+S}(t) \tag{19}$$

C is an input parameter that adjusts the level of semitones on either side of the individual harmonic outputs. The matrices are offset by $\pm S$ since there are 14 modes per semitone. I simply shifted copies of a'_w vertically and padded with zeros to implement this effect. $a''_w(t)$ then replaced $a'_w(t)$ in equation (8).

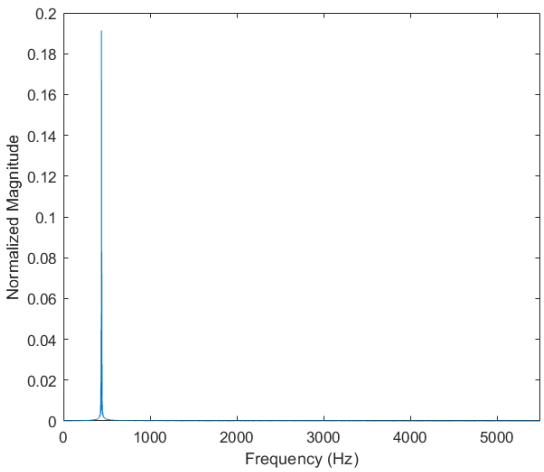
4 Results and Discussion

In this section, I will present the results of my implementation in MATLAB. In all of the examples shown, a sinusoidal waveform was used as input. For various drawbar and crosstalk settings, the frequency response of each output clearly shows the corresponding pseudo-harmonic gains relative to the input waveform. I will show the frequency response of these effects separately and in tandem with one another.

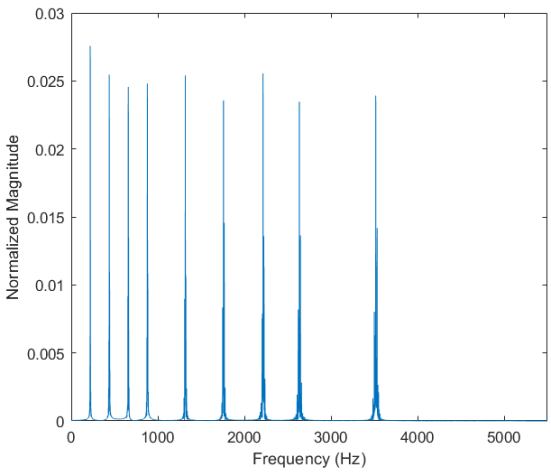
4.1 Drawbar Control Examples

Based on a 1.5 second 440 Hz sine wave input, Figure 5 shows the frequency response of two different registrations, [008000000] and [888888888], outputting the fundamental and all pseudo harmonics respectively. The crosstalk level, C, is set to zero for both. For reference, the semitone offsets assigned to each drawbar are found in (14).

Figure 5. 1.5 second sine wave input at 440 Hz



(a) Registration: [008000000]



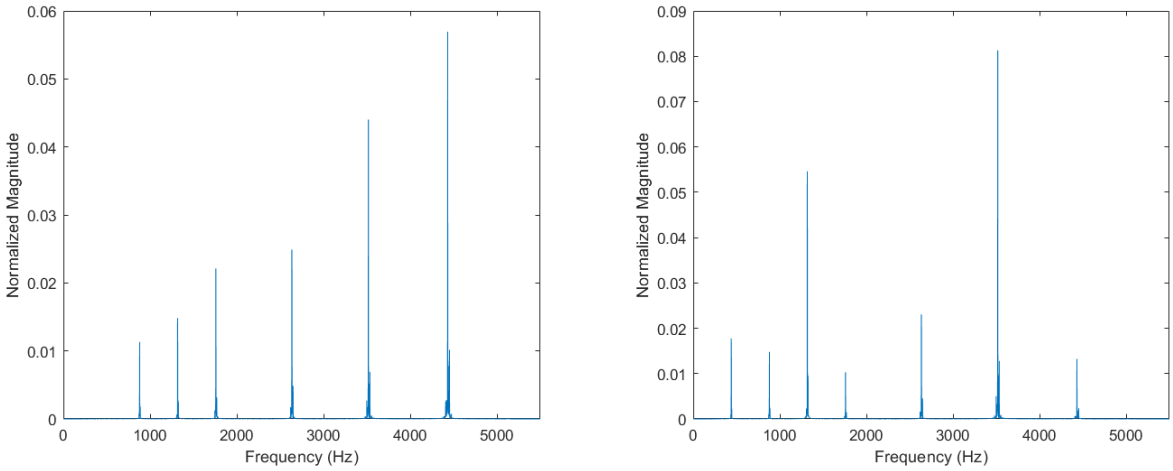
(b) Registration: [888888888]

The fundamental is clearly visible in 5a, at 440 Hz. In 5b, nine total magnitudes representative of each pseudo-harmonic can be seen. There is slight variability in the

magnitude of each, which I was unable to fully mitigate. I have verified that each of the offsets in 5b are in the correct position based on the input frequency.

Figure 6 is based on a 1.5 second 880 Hz sine wave with two new registrations, [021345678] and [473258300].

Figure 6. 1.5 second sine wave input at 880 Hz



(a) Registration: [021345678]

(b) Registration: [473258300]

6a has no presence from the octave below, and the rest of the harmonics increase almost linearly. The second offset gain value corresponds to +7 semitones above the fundamental, which is in the third gain position. Only six distinct harmonics appear on the frequency response since the highest two are above the 5120 Hz cap to the implementation's range ($(880)2^{31/12} = 5274$ Hz, $(880)2^{36/12} = 7040$ Hz).

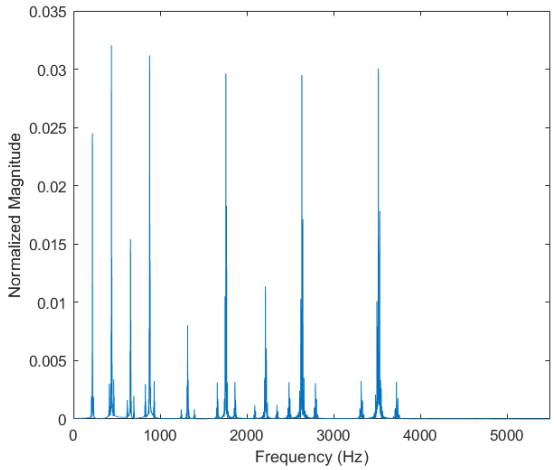
6b features a set of varying harmonic gains. The drawbar controls work as expected, allowing for individual control over each harmonic, with relative magnitudes representative of each expected frequency.

4.2 Crosstalk Examples

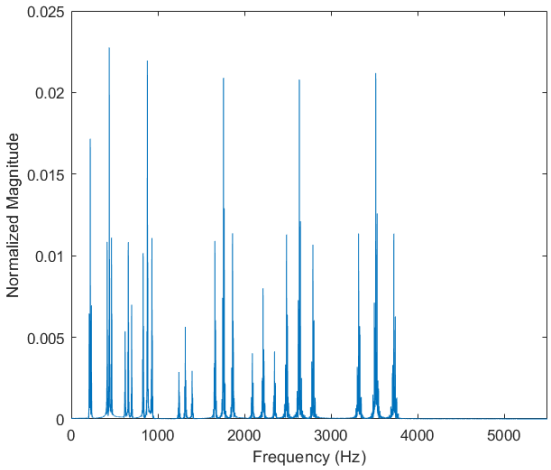
The crosstalk level, $C \in [0, 1]$, is an easily adjustable parameter that controls the presence of ± 1 semitonal offsets of each harmonic in the Hammond effect output. Figure 7 shows two separate crosstalk value settings, $C = 0.1$, and $C = 0.5$. The registration for both plots is [668848588] on a 1.5 second 440 Hz sin wave input.

The crosstalk gain is relative to each harmonic's individual drawbar setting. In 7b, notice how the levels off the semitone offsets are at about half the magnitude of their respective harmonic. Neither frequency responses are audibly pleasurable, so I usually set $C \leq .04$.

Figure 7. 1.5 second sine wave input at 440 Hz with [668848588] registration



(a) C = 0.1



(b) C = 0.5

4.3 Perceptual Output

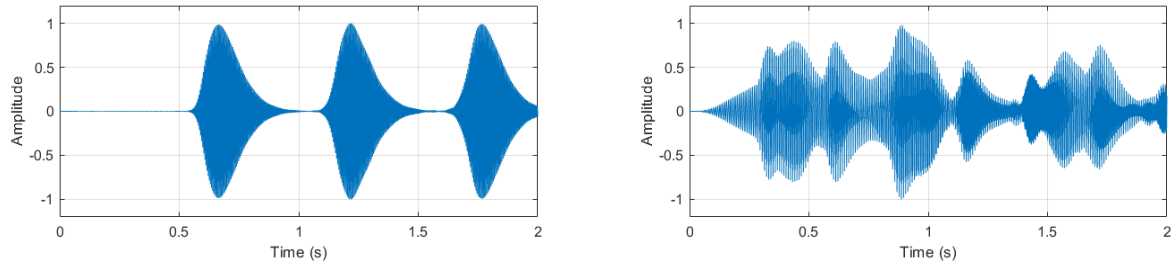
Listening to various drawbar registrations and crosstalk levels, the Hammond Effects synthesis was partially successful. Registrations only featuring the fundamental or single offsets, such as [008000000] or [000008000], led to an output with distinct tremolos of similar length. As the sole offset frequencies increased, the lengths of the tremolos shortened.

The tremolos were caused by the double filtering in $g_{post}(t)$, which also acts as an anti-aliasing filter. When I remove the filter, no tremolo is perceived, but a high to low frequency sweep is more present. It was difficult to fully reduce any aliasing, as I was not expecting it for any of the sinusoidal inputs tested. The modulation steps in equations (5) and (7) offset by at most 5120 Hz, which does not cause a shift close to the Nyquist frequency for the example inputs of 440 and 880 Hz. I also checked the output with and without a high-order Butterworth filter, though the sweep is still perceivable.

The tremolo effect is frequency dependent. At 220 Hz, separate attacks are perceived on a 2 second interval with fundamental registration [008000000]. As the frequency increases, the time between amplitude notches decreases, and the initial transient adjusts earlier in time. Successive tremolo envelopes begin to overlap as well, leading to a less present effect. Once there is enough overlap, the tremolo speeds up again with much more distinct notches.

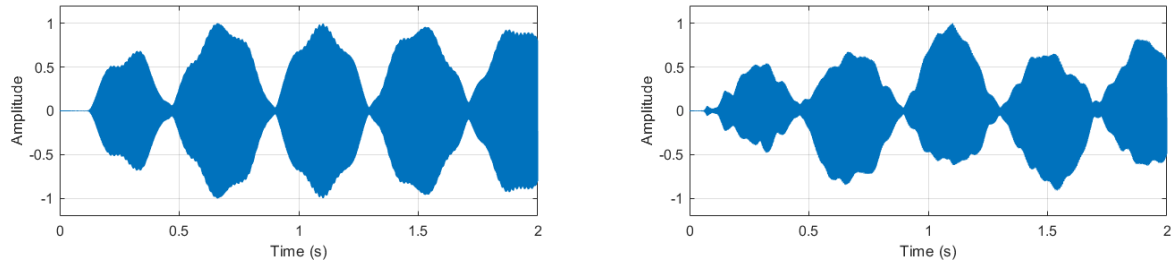
Thankfully, the tremolo is a welcome effect, since earlier versions of the Hammond were routed into a Leslie speaker, which caused a tremolo on the organ's output. Figure 8 and Figure 9 both show two waveforms with varied registrations.

Figure 8. 2 second sine wave input at 220 Hz



(a) Registration: [008000000] **(b)** Registration: [846280000]

Figure 9. 2 second sine wave input at 1100 Hz



(a) Registration: [008000000] **(b)** Registration: [846280000]

5 Conclusion

Based on Werner and Abel's paper, I was able to successfully implement a subset of the effects processing they had suggested for the Hammond organ. The modal reverberator architecture was augmented to perform a series of filtering and matrix multiplication operations in MATLAB for sinusoidal inputs of varying length and frequency. With 1177 modes to compute across, the task provided to be very computationally demanding for the realistic frequency range of the organ. These modes can be easily scaled accordingly, though increasing may induce more unwanted artifacts.

Generally speaking, each step in the architecture was successfully executed. My model heterodynes (modulates) and demodulates, filters across every mode, adjusts pseudo harmonic magnitudes, introduces additional crosstalk, and adds a tremolo into the processed sound.

The drawbar matrix allows for each harmonic to be individually altered, which serves as the core functionality of the Hammond. Nine drawbars with nine positions each leads to a widely adjustable output spectrum. Crosstalk is induced easily through the augmentation of a single parameter, which can be tweaked to mimic the organ's timbre. Surprisingly, filtering operations contributed to a frequency-dependent tremolo effect that added a pleasant layer to the perceptual output. My implementation only shined for sinusoidal inputs, though they are the most reminiscent of the Hammond's characteristic sound.

References

[1] Abel, J.S.; Coffin, S.; Spratt, K.S. A modal architecture for artificial reverberation with application to room acoustics modeling. In Proceedings of the 137th Convention of the Audio Engineering Society (AES), Los Angeles, CA, USA, 9–12 October 2014.

[2] Abel, J.S.; Werner, K.J. Distortion and pitch processing using a modal reverb architecture. In Proceeding of the 18th International Conference on Digital Audio Effects (DAFx-15), Trondheim, Norway, 30 November–3 December 2015.

[3] Pekonen, Jussi & Pihlajamäki, Tapani Välimäki, Vesa. (2011). Computationally efficient Hammond organ synthesis. Proceedings of the 14th International Conference on Digital Audio Effects, DAFx 2011.

[4] Werner, K.J.; Abel, J.S. Modal Processor Effects Inspired by Hammond Tonewheel Organs. *Appl. Sci.* **2016**, 6, 185.

[5] Werner, K.J.; Abel, J.S. "Modal Processor Effects Inspired by Hammond Tonewheel Organs"—Audio Examples. Available online: https://ccrma.stanford.edu/ kwerner/appliedsciences/hammondizer.html (accessed on 9 October 2018).

Appendices

A Hammondizer Code

```
% Evan Savage - Hammondizer script
% By adjusting the crosstalk level and drawbar controls, one can simulate
% the effects of the hammond organ. Useful parametes to set are:
%
% fc - sinusoidal input frequency
% duration - time of output file (setting it > 2 is unfortunately very
%            computationaly demanding
% C - amount of crosstalk from one semitone away ( > .1 makes for a very
%     unsettling sound)
% draw_pos - the respective level of each drawbar, the harmonic offsets of
%            each can be seen in the 'o_T' array

clear;
clf;

fs = 44100;              % sampling rate
Ts = 1/fs;               % discrete time sample
duration = 2;            % duration (s) of input
t = 0:Ts:duration-Ts;    % array of discrete time sample indices

fc = 440;                % frequency of sine wave input

x = cos(2*pi*fc*t);      % sine wave input

S = 14;                  % modes per semitone
C = .01;                 % crosstalk level
alpha = 0.0003;          % g_post coefficient
modes = 1177;            % 7 octaves * 12 semitones * 14 modes/semitone

% gpre feedforward coefficients (feeback is just 1)

gpre = (1/modes)*ones(1,modes);

% g_post filter coefficients

b0 = 0;
b1 = alpha*exp(-alpha);
```

```
a0 = 1;
a1 = -2*exp(-alpha);
a2 = exp(-2*alpha);

% Set individual drawbar positions to increments of -3 (dB) besides 0

% pos   [    0      1      2      3      4      5      6      7      8]
% ------------------------------------------------------------------
% gain [-inf    -21    -18    -15    -12     -9     -6     -3      0]

draw_pos = [0 2 1 3 4 5 6 7 8];

% Convert drawbar increments (dB) to amplitudes

draw_val = zeros(1,9);
ind = 1;

for pos = draw_pos
    if pos == 0
        draw_val(ind) = 0;
    else
        draw_val(ind) = 10.^(((draw_pos(ind)-8)*3)/20);
    end

    ind = ind + 1;
end

% Semitonal offsets for each drawbar

o_T = [-12;7;0;12;19;24;28;31;36];

% Routing matrix
% - "kron.m" includes the Kronecker delta function: kron(x)
% - Sparsely filled with at most 9 diagonals for each drawbar

routing_mat = zeros(modes, modes);

for w = 1:modes
   for k = 1:modes
      for d = 1:9
        routing_mat(w,k) = routing_mat(w,k) ...
            + draw_val(d)*kron(w-k-o_T(d)*S);
      end
```

```
    end
end

% Range of mode frequencies exponentially spaced across the organ's range

f_w = zeros(modes,1);

for z=1:modes
    f_w(z) = 40*2^(z/(12*S));
end

% Convert to radial frequencies

omega_w = 2*pi*f_w;

% Heterodyning sinusoids

phi = cos(omega_w * t);

% Demodulating waveforms, lowest octave are square

psi = zeros(modes, length(t));

for w = 1:modes
    if w < S*12
        psi(w,:) = 4/pi*cos(omega_w(w)*t) ...
            + 4/(3*pi)*cos(3*omega_w(w)*t) ...
            + 4/(5*pi)*cos(omega_w(w)*t);
    else
        psi(w,:) = cos(omega_w(w)*t);
    end
end

% Filter the modulated input with a moving average (lowpass) filter

p = filter(gpre, 1, phi .* x, [], 2);

% Multiply with routing matrix to apply pseudo harmonic gains

amp = routing_mat * p;

% Filter twice through with g_post
```

```
amp_p = filter([b0 b1], [a0 a1 a2], amp, [], 2);
amp_p = filter([b0 b1], [a0 a1 a2], amp_p, [], 2);

% Crosstalk matrix, offset amp_p by +/- 1 semitone with crosstalk level, C

amp_pp = C*[amp_p(S+1:end, :); zeros(S, length(t))] ...
    + amp_p + C*[zeros(S, length(t)); amp_p(1:end-S, :)];

% Demodulate to output matrix

y = psi .* amp_pp;
N = length(y);

% sum all rows of y for output vector and normalize

out = sum(real(y));
out = real(out)/max(abs(real(out)));

% Plot of output waveform

figure(1)
plot(t, real(out));
axis([0, duration, -1.2, 1.2]);
xlabel('Time (s)');
ylabel('Amplitude');
grid

% Frequency response of output waveform

figure(2)
f_x = [0:N-1] * fs / N;

F = abs(fft(out))/ N;

plot(f_x, F);
xlim([0 5500]);
xlabel('Frequency (Hz)');
ylabel('Normalized Magnitude');
```

B Kronecker Delta Function Code

```
function out = kron(x)
    if x == 0
        out = 1;
    else
        out = 0;
    end
end
```