Abstract: This paper details the steps taken to implement a variety of effects with a phase vocoder. After implementing the effects in Matlab, a simple easy-to-use interface was designed to allow a user to more easily alter sound and to make their own digital audio effects.

The Matlab vocoder was implemented using techniques found in Traditional Implementations of a Phase-Vocoder: The Tricks of the Trade, by Amalia De Götzen, Nicola Bernardini, and Daniel Arfib [1]. After designing the vocoder, the functionality of the vocoder was tested with several different audio inputs. Each of the following effects were created and implemented with Matlab.

1. INTRODUCTION

1.1. Phase-Vocoders

The phase vocoder is a widespread device, used to implement a wide variety of digital audio effects, such as time-scaling and frequency shifting, as well a number of voice distorting effects like robotization and whisperization. There are multiple ways to implement the phase vocoder, each with its own advantages.

1.2. Implementation Techniques

1.2.1 Direct FFT/IFFT Approach

The direct FFT/IFFT approach to phase vocoding is implemented in three key stages, analysis, processing, and resynthesis. During the analysis stage, the short-time Fourier transform (STFT) is applied to the input signal to transfer it into the frequency domain. This is done by windowing the input signal using an appropriate windowing function (often a Hanning window). A series of windows are overlapped on top of the input signal, with an appropriate time delay between each (hop size). When choosing the hop size, one must take care to not choose a delay that is too large as to not accurately represent the input signal, but not so small that it requires too much time to process. After the input has been windowed, the fast Fourier transform (FFT) is applied to each section of the windowed input (bin).

It is during the processing stage that the frequency representation of the signal is modified in order to add audio effects. The implementation of some of the possible effects is covered at a later point in this paper. After the processing stage, the now modified signal needs to be reconstructed into the time-domain. This is done using the same techniques as in the analysis stage, only the operations are reversed. The inverse FFT is applied to the segments, after which
they are sequentially added and overlapped to produce a time-domain representation of the modified signal. If the input signal is not changed during the processing stage, the output after resynthesis is a similar copy of the input.

1.2.2 Filter Bank Approach

In this method, the input signal is treated as a sum of sinusoids, each of which is modulated in amplitude and frequency. As the signal is input into the vocoder, a bank of filters separates the signal into a series of several filtered versions. Each filtered version is then transformed into the frequency domain where magnitude and phase processing is done to apply effects to the signal. Each filtered version is then resynthesized into the time domain and summed to obtain the new, modified signal.

Figure 1: Direct FFT/IFFT Approach

Figure 2: Filter Bank Implementation

2. PHASE-VOCODER EFFECTS

2.1. Time Stretching

Time shifting without changing pitch is one of the most common effects implemented with phase vocoders. Despite being one of the most difficult digital audio effects to execute, it is relatively simple with phase vocoders. In Improved Phase Vocoder Time-Scale Modification of Audio [2], Laroche and Dolson describe a technique for more efficient time stretching; one that reduces the “phasiness” that is often present in vocoders, as well as decreasing the computational cost of time-shifting. Laroche and Dolson present two cases that occur when using this technique; when the time stretching factor is an integer multiple of frequency bins, and when it is not. The former case substantially decreases the computations necessary to time shift, and also does not need a phase-unwrapping technique. It does however, not work as effectively for signals with a high sampling rate or a small FFT size.

As with all effects, this method of time-shifting occurs in the processing phase. After the input signal has been windowed and each bin has been transformed into the frequency domain, they are recompiled into the time by using a different hop size. This change in
hop size stretches or compresses the time-domain representation of the signal.

2.2. Pitch Shifting

After performing the FFT on the input signal, each window is time stretched in the frequency domain (as before) and then resampled using the inverse of the time stretching ratio. These new bins are overlapped and added to obtain the spectrum of the new signal, which is then transformed into the time domain using the IFFT.

2.3. Robotization & Whisperization

The robotization effect applies a fixed pitch to the output sound, forcing the sound to be periodic. This is a very simple effect to apply, as it simply involves setting the phase of each FFT to a constant value before the signal is resynthesized. Whisperization is a similar effect, except, instead of setting the phase of the FFTs to zero; it sets them to random values between 0 and $2\pi$.

2.4. Denoising

Denoising is an effect that will remove or reduce unwanted low amplitude sound on audio signals. These unwanted sounds include background noise, artifacts that have been produced by previously applied nonlinear operations, or to emphasize some specific frequencies. After the analysis stage, when the magnitude and phase of all frequency bins are known. Unlike with robotization and whisperization where the phase of the input signal is changed, with denoising, the magnitude of the frequency representation is changed such that all sound below a threshold is attenuated.

2.5. Stable/Transient Components Separation

Stable/transient components separation involves separating so-called “stable” sounds from “transients.” Transients are short-duration signals that contain a large degree of non-periodic components, similar to drum beats or hand claps. This is done in a vocoder by calculating the instantaneous frequency of a bin by differentiating phase with respect to time. After the instantaneous frequency is known, it is possible to classify the sound as stable or transient by checking to see if the instantaneous frequency is within preset bounds. Once classified, the bin is either used in the signal reconstruction or is discarded.

2.6. Mutation between Two Sounds

This effect involves calculating a frequency representation from two input signals which is then transformed into an output sound based on the two inputs. Figure 3 shows approximately how the two signals are combined. Firstly, the magnitude and phase components of both input signals are found using a FFT. After both sets of magnitude and phase are known, they can be multiplied or added together to obtain several different effects [6].

![Figure 3: Mutation between Two Sounds](image)

3. CONCLUSIONS
As is shown above, phase vocoders are a useful tool in the arsenal of an audio engineer. With them it is possible to create a large number of digital audio effects, and it allows for easy creation of more. There do exist problems with the vocoder, however, the biggest of which is phasiness. There are numerous methods described in IEEE journals that explain algorithms and processes to reduce phasiness, but even the best of them has its drawbacks, either the algorithms are computationally intense, or they simply do not work on signals of certain sampling rates.

4. REFERENCES


function varargout = newgui(varargin)

% NEWGUI M-file for newgui.fig
%   NEWGUI, by itself, creates a new NEWGUI or raises the existing
%   singleton*.
%   H = NEWGUI returns the handle to a new NEWGUI or the handle to
%% the existing singleton*.
%% NEWGUI('CALLBACK',hObject,eventData,handles,...) calls the local function named CALLBACK in NEWGUI.M with the given input arguments.
%% NEWGUI('Property','Value',...) creates a new NEWGUI or raises the existing singleton*. Starting from the left, property value pairs are applied to the GUI before newgui_OpeningFunction gets called. An unrecognized property name or invalid value makes property application stop. All inputs are passed to newgui_OpeningFcn via varargin.
%% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".
%% See also: GUIDE, GUIDATA, GUIHANDLES

% Last Modified by GUIDE v2.5 20-Jul-2008 23:58:08

%% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                    'gui_Singleton',  gui_Singleton, ...
                    'gui_OpeningFcn', @newgui_OpeningFcn, ...
                    'gui_OutputFcn',  @newgui_OutputFcn, ...
                    'gui_LayoutFcn',  [] , ...
                    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
%% End initialization code - DO NOT EDIT

%% --- Executes just before newgui is made visible.
function newgui_OpeningFcn(hObject, eventdata, handles, varargin)
%% This function has no output args, see OutputFcn.
%% hObject handle to figure
%% eventdata reserved - to be defined in a future version of MATLAB
%% handles structure with handles and user data (see GUIDATA)
%% varargin command line arguments to newgui (see VARARGIN)

%% Choose default command line output for newgui
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes newgui wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = newgui_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in open.
function open_Callback(hObject, eventdata, handles)
% hObject    handle to open (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

switch get(handles.implementation, 'Value')
    case 1
        %------File Opening----------------
        [filename, pathname] = uigetfile('*.wav', 'Pick a wav-file');
        if isequal(filename,0)
            disp('User selected Cancel')
        else
            file = fullfile(pathname, filename);
        end
        set(handles.filennamelabel,'String',
             ['Filename: ' pathname filename]);

        %------Algorithm Begins-----------
        WLen = str2double(get(handles.chunk, 'String'));
        steps_1 = str2double(get(handles.steps, 'String'));
        stretchratio = str2double(get(handles.stretchratio, 'String'));
        steps_2 = stretchratio * steps_1;

        %------Windowing Techniques------
        switch get(handles.windowpopup, 'Value')
            case 1
                % this is hann
                w1 = hann(WLen);
            case 2
                % this is hamming
                w1 = hamming(WLen);
case 3
% this is blackman
set(handles.windowpopuptext,'String','Window Method: blackman');
w1 = blackman(WLen);

w1 = blackman(WLen);
case 4
% this is flattop
set(handles.windowpopuptext,'String','Window Method: flattop');

w1 = flattopwin(WLen);
case 5
% this is chebyshev
set(handles.windowpopuptext,'String','Window Method: chebyshev');

w1 = chebwin(WLen);
end

[input, FS] = wavread(file);

L = length(input);
input = [zeros(WLen,1);input;zeros(WLen,1)]/ max(abs(input));
%the appended zeros to the back of the input sound file makes it so that the windowing samples the complete sound file

%----- initializations -----
output = zeros(WLen + ceil(length(input)* stretchratio),1); count_in  = 0; count_out  = 0;

while count_in<(length(input) - WLen)
grain = input(count_in+1:count_in+WLen).* w1;
%fILTERING!
switch zrp
    case 1
        % this is no filter on
    case 2
        % this is robot!
            phi = 0;
    case 3
        % this is whisper.
            phi = 2*pi*rand(WLen,1);
    case 4
        % this is denoising.
            denoise = str2double(get(handles.denoise, 'String'));
            if r >= 0.001
                ft = f.*(r./(r+denoise));
            else
                ft = f.*(r./(r+sqrt(denoise)));
            end
end
if zrp ~= 4,
    ft = ceil((r.* exp(i*phi)));
end
grain = real(ifft(ft)).*w1;

% output(count_out+1:count_out+WLen) =
output(count_out+1:count_out+WLen) + grain;
    count_in  = count_in + steps_1;
    count_out  = count_out + steps_2;
end

% == Save File ================================
[filename, pathname] = uiputfile('*.wav', 'Save As...');
if isequal(filename,0) | isequal(pathname,0)
    disp('User selected Cancel')
else
    file = fullfile(pathname,filename);
    disp(['User selected: ',file])
end

output = output(1:length(output)) / max(abs(output));

plot(handles.axes1,input);
plot(handles.axes2,w1);
plot(handles.axes3,abs(fftshift(fft(w1))));
plot(handles.axes4,output);

if (zrp == 5)
    FS = FS * stretchratio;
end
wavwrite(output, FS, file);
case 2

% filterbank.m

%-------File Opening----------------
[filename, pathname] = uigetfile('*.wav', 'Pick a wav-file');
if isequal(filename,0)
    disp('User selected Cancel' )
else
    file = fullfile(pathname, filename);
end
set(handles.filenameLabel,'String',['Filename: ' pathname filename]);

WLen = str2double(get(handles.chunk, 'String'));
NChannel = str2double(get(handles.nchannel, 'String'));

n1 = 4*WLen;
[input,FS] = wavread(file);
L = length(input);

switch get(handles.windowpopup, 'Value')
case 1
  % this is hann
  set(handles.windowpopuptext,'String','Window Method: hann');
  w1 = hann(WLen);
case 2
  % this is hamming
  set(handles.windowpopuptext,'String','Window Method: hamming');
  w1 = hamming(WLen);
case 3
  % this is blackman
  set(handles.windowpopuptext,'String','Window Method: blackman');
  w1 = blackman(WLen);
case 4
  % this is flattop
  set(handles.windowpopuptext,'String','Window Method: flattop');
  w1 = flattopwin(WLen);
case 5
  % this is chebyshev
  set(handles.windowpopuptext,'String','Window Method: chebyshev');
  w1 = chebwin(WLen);
end

output = zeros(length(input),1);
X_tilde = zeros(n1,nChannel);
z = zeros(WLen-1,nChannel);

t = (-WLen/2:WLen/2-1)';
ourFilter = zeros(WLen, nChannel);
for k=1:nChannel,
    wk = 2*pi*i*(k/WLen)';
    ourFilter(:,k) = w1.*exp(wk*t);
end

p = 0;
pend = length(input) - n1;

while p < pend,
    grain = input(p+1:p+n1);
    for k=1:nChannel,
        [X_tilde(:,k),z(:,k)] = filter(ourFilter(:,k),1,grain,z(:,k));
    end
    res = real(sum(X_tilde,2));
    output(p+1:p+n1) = res;
    p = p + n1;
end
output = output / max(abs(output));

plot(handles.axes1,input);
plot(handles.axes2,w1);
plot(handles.axes3,abs(fftshift(fft(w1))));
plot(handles.axes4,output);

[filename, pathname] = uiputfile('*.wav', 'Save As...');
if isequal(filename,0) | isequal(pathname,0)
    disp('User selected Cancel')
else
    file = fullfile(pathname,filename);
    disp(['User selected: ',file])
end

wavwrite(DAFx_out,FS,file);

end

% --- Executes on selection change in implementation.
function implementation_Callback(hObject, eventdata, handles)
% hObject    handle to implementation (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns implementation
% contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
% implementation

% --- Executes during object creation, after setting all properties.
function implementation_CreateFcn(hObject, eventdata, handles)
% hObject    handle to implementation (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function nchannel_Callback(hObject, eventdata, handles)
% hObject    handle to nchannel (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of nchannel as text
%       str2double(get(hObject,'String')) returns contents of nchannel
% as a double

% --- Executes during object creation, after setting all properties.
function nchannel_CreateFcn(hObject, eventdata, handles)
% hObject    handle to nchannel (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
  get(0,'defaultUicontrolBackgroundColor'))
  set(hObject,'BackgroundColor','white');
end