Prof. Dr. Stefan Weinzierl 16.01.2013

Musterlösung: 22. Januar 2014, 16:51

Konvertieren von bereits bestehenden Effekten in Blockverarbeitungs-kompatible Versionen derselbigen

Im Tutorium haben wir folgende Effekte in Matlab implementiert: FIR-Kammfilter, IIR-Kammfilter, universeller Kammfilter (Kombination aus FIR- & IIR-Kammfilter) sowie den Vibrato-Effekt (geringe Modulation der Tonhöhe). Da wir uns bisher allerdings noch keine Gedanken zur Implementierung in unser Blockverarbeitungsschema Gedanken gemacht haben, soll dies nun an dieser Stelle geschehen.

a) Erstellen Sie eine Blockverarbeitungs-Version des unicomb.m-Effekts aus dem Tutorium.

Lösung:

```
function [ output, buffer ] = unicombBlock( signal, Fs, delay, FF, FB, BL, buffer )
%UNICOMB Comb-filter as a combination of a FIR- and an IIR-comb-filter
% signal: input signal (column vector)
% Fs    : sampling frequency of the input signal
% delay : delay between original and delayed version of input in seconds
% FF    : FeedForward factor for the delayed input sample
% FB    : FeedBack factor for the delayed ouput sample
% BL    : BLend factor for the "dry" signal
%
% Taken and edited from U. Zoelzer "DAFX - Digital Audio Effects", 2002,
% Wiley

% allocate memory for output:
output = zeros(size(signal));

% convert delay from seconds to number of samples
taps = round(delay*Fs);

% initialize buffer if none is given:
if nargin==6
    % initialize delay line
    buffer = zeros(taps,1);
end

for n=1:length(signal)
    xh = signal(n) + FB * buffer(taps);
    output(n) = FF * buffer(taps) + BL * xh;
    buffer = [xh;buffer(1:taps-1)];
end

end
```

b) Erstellen Sie das gleiche für den vibrato.m-Effekt.

Lösung:

```
function [output, buffer, phi]=vibratoBlock(signal,Fs,Modfreq,Width,Delay,buffer,phi)
% VIBRATO calculates a vibrato effect for input signal x
% INPUT:
% signal    -   mono input signal
% Fs        -   sampling-frequency of the input signal
% Modfreq   -   modulation frequency in Hz (typically: 5-14 Hz)
% Width     -   width of the Modulation in ms (typically: 5-10 ms)
% optional:
% Delay     -   basic delay in ms, has to be >= Width!
% buffer    -   past values of x (x[n-1] etc.)
% phi       -   phase of modulation signal
%
% OUTPUT:
% output    -   mono output signal
% buffer    -   past values of x (x[n-1] etc.)
% phi       -   phase of modulation signal
%
% Taken and edited from U. Zoelzer "DAFX - Digital Audio Effects", 2002,
% John Wiley and sons

% # of samples in input
N = length(signal);
% memory allocation for output vector
output = zeros(size(signal));

% modulation width in # samples:
WIDTH = round(Width*Fs);

% if a Delay is given it is converted from seconds to # samples. If no
% Delay is given it is set to Width (smallest possible delay).
if nargin>5
    DELAY = round(Delay*Fs);
    L = length(buffer);
elseif nargin==5
    % basic delay in # samples:
    DELAY = round(Delay*Fs);
    % initialize delay line:
    % length of the entire delay
    L=2+DELAY+WIDTH*2;
    % memory allocation for delay
    buffer=zeros(L,1);
    % initialize phase:
    phi = 0;
elseif nargin==4
    DELAY = round(Width*Fs);    % basic delay in # samples
    % initialize delay line:
    % length of the entire delay
    L=2+DELAY+WIDTH*2;
    % memory allocation for delay
    buffer=zeros(L,1);
    % initialize phase:
    phi = 0;
end


% WIDTH has to be greater than DELAY for the correct memory allocation of
% the delayline
if WIDTH>DELAY
  error('modulation width greater than basic delay !!!');
```

```
return;
end

% modulation frequency in # samples
MODFREQ = Modfreq/Fs;

for n=1:N
    % shift buffer one sample:
    buffer=[signal(n);buffer(1:L-1)];

    % modulation, exchange with LP-filtered white noise for other effects:
    MOD=sin(MODFREQ*2*pi*n+phi);

    % sample# of the playback sample to be (non-integer):
    ZEIGER=1+DELAY+WIDTH*MOD;

    % Linear Interpolation
    % ----------------------
    % find nearest integer sample smaller than ZEIGER
    i=floor(ZEIGER);
    frac=ZEIGER-i;
    output(n,1)=buffer(i+1)*frac+buffer(i)*(1-frac);
end

phi = rem(MODFREQ*2*pi*n+phi, 2*pi);
```

c) Erzeugen Sie ein Host-Skript, welches die Funktionsweise einer DAW simuliert. Das heißt ein Skript, dass eine Mono-Datei einliest und in Blöcken der Länge N an ihre in **a)** und **b)** geschriebene Plugins weitergibt. Als Audiodateien können Sie die hochgeladenen Beispieldateien benutzen. Erzeugen Sie für mindestens ein Eingangssignal jeweils ein Ausgangssignal für Filtereinstellungen ihrer Wahl für Vibrato und Combfilter und prüfen Sie ob Sie den gleichen Output bei blockweiser Verarbeitung und nicht-blockweiser Verarbeitung herauskriegen.

Lösung:

```
%% Aufgabe c

clear all
close all
clc

% Einlesen des zu filternden Signals
[signal,Fs] = audioread('Sprache.wav');
signal = signal(:,1);
temp_signal = signal;

% Festsetzen der Blocklaenge:
N = 512;

% Berechnen wieviele Bloecke der Laenge N aus dem Eingangssignal erstellt
% werden koennen und falls die Anzahl der Bloecke nicht ganzzahlig ist,
% erweitern mit Nullen:
numFrames = ceil(length(signal)/N);
if mod(length(signal),N)~= 0
    numNull = N-mod(length(signal),N);
    temp_signal = [temp_signal;zeros(numNull,1)];
end

```

```
%% Vibrato
% Parameter fuer Vibrato festlegen:
fMod = 5;    % Modulationsfrequenz in Hz
M = 5e-3;    % Statisches Delay in Sekunden
A = 5e-3;    % Modulationstiefe in Sekunden

% Berechnen des Outputs in Bloecken:
for idx = 1:numFrames
    % Frame aus dem Gesamtsignal herausloesen:
    bufferIn = temp_signal((idx-1)*N+1:idx*N,1);

    if  idx == 1
        [bufferOut,buffer,phi] = vibratoBlock(bufferIn,Fs,fMod,A,M);
    else
        [bufferOut,buffer,phi] = vibratoBlock(bufferIn,Fs,fMod,A,M,...
            buffer,phi);
    end

    output_vib((idx-1)*N+1:idx*N,1) = bufferOut;
end

% Kuerzen des Outputs um die am Anfang angehaengten Nullen
output_vib = output_vib(1:end-numNull);

% Berechnen des Ouputs ohne Blockweise Verarbeitung:
output_vib_no_block = vibrato(signal,Fs,fMod,A);

% Vergleich plotten:
diff_vibr = abs(output_vib-output_vib_no_block);
figure;
plot(diff_vibr);
title('Differenz aus blockweiser und konventioneller Verarbeitung',...
    'Fontsize',14);
xlabel('Samplenummer','Fontsize',14);
ylabel('abs(output_{blockweise}-output_{nicht-blockweise})',...
    'Fontsize',14);

%% Unicomb

% Parameter fuer Unicomb festlegen:
delay = 10e-3;  % Delay zwischen Original und Kopie in Sekunden
FF = 0;         % FeedForward-Faktor, verzoegertes direkt gemischtes Sample
FB = 0.3;       % FeedBack-Faktor, verzoegertes, zurueckgefuehrtes Sample
BL = 0.7;       % BLend-Faktor, unveraendertes Eingangssignal

% Berechnen des Outputs in Bloecken:
for idx = 1:numFrames
    % Frame aus dem Gesamtsignal herausloesen:
    bufferIn = temp_signal((idx-1)*N+1:idx*N,1);

    if  idx == 1
        [bufferOut,buffer] = unicombBlock(bufferIn,Fs,delay,FF,FB,BL);
    else
        [bufferOut,buffer] = unicombBlock(bufferIn,Fs,delay,FF,FB,BL,...
            buffer);
    end

    output_comb((idx-1)*N+1:idx*N,1) = bufferOut;
end

```

```
84 % Kuerzen des Outputs um die am Anfang angehaengten Nullen
85 output_comb = output_comb(1:end-numNull);
86
87 % Berechnen des Ouputs ohne Blockweise Verarbeitung:
88 output_comb_no_block = unicomb(signal,Fs,delay,FF,FB,BL);
89
90 % Vergleich plotten:
91 diff_comb = abs(output_comb-output_comb_no_block);
92 figure;
93 plot(diff_comb);
94 title('Differenz aus blockweiser und konventioneller Verarbeitung',...
95     'Fontsize',14);
96 xlabel('Samplenummer','Fontsize',14);
97 ylabel('abs(output_{blockweise}-output_{nicht-blockweise})',...
98     'Fontsize',14);
```