Computer Music in Undergraduate Digital Signal Processing

Phillip L. De Leon
New Mexico State University
Klipsch School of Electrical and Computer Engineering
Las Cruces, New Mexico 88003-800
pdeleon@nmsu.edu

Abstract

In a typical undergraduate treatment of Digital Signal Processing (DSP), waveform synthesis (digital creation of signals) is often not covered for the sake of allocating time to other applied material such as digital filter design and spectral analysis. While basic digital filtering theory (difference equations) is fairly elementary and understood by students early on in a typical course, students are not given much opportunity for the application to waveform synthesis. In this paper we outline a waveform synthesis project in which students code two simple tools in MATLAB. These tools are a tone synthesizer and an envelope generator used to shape the amplitude of the tone, i.e. amplitude modulation. These two tools form the basis for a waveform synthesis project where students can experiment with computer-based music and musical synthesis using MATLAB's built-in sound capabilities and the PC's sound card. A student design example is provided which will synthesize a Bach mussette (25 seconds in duration) using only the above tools in MATLAB. Experience in digital waveform synthesis, can provide a basis for application to other engineering solutions which require waveform or signal synthesis such as data communications devices (modems), software radios, and DTMF (Touch Tone) generators.

1. Introduction

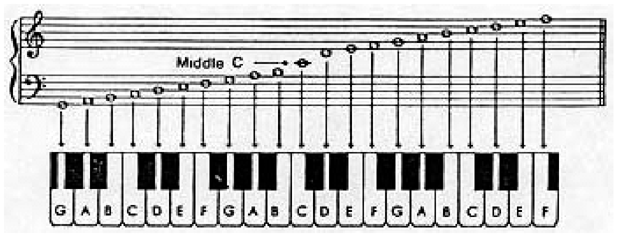
The ability to synthesize waveforms through digital methods is a popular technique. This method can be found in many applications such as data communications devices (modems), software radios, and DTMF (Touch Tone) generators. One of its most familiar consumer-oriented applications is in music synthesis. In this application, the musician often has control over many instruments and sound effects all from a single synthesizer. Waveform synthesis can be taught early in a typical undergraduate Digital Signal Processing (DSP) course to illustrate some of the applications of sampling and reconstruction theory. In addition hands-on practice with waveform synthesis can be made very interesting in the context of computer music. In this paper we outline a waveform synthesis project in which students code two simple tools in MATLAB. These tools are a tone (sinusoid) generator and an envelope generator used to shape the amplitude of the tone, i.e. amplitude modulation. These two tools form the basis of the project where students can experiment with computer-based music and musical synthesis using MATLAB's built-in sound capabilities and the PC's sound card. A student design example is provided which will synthesize a Bach mussette (25 seconds in duration) using only the tone and envelope generation tools in MATLAB.

2. Music Synthesis

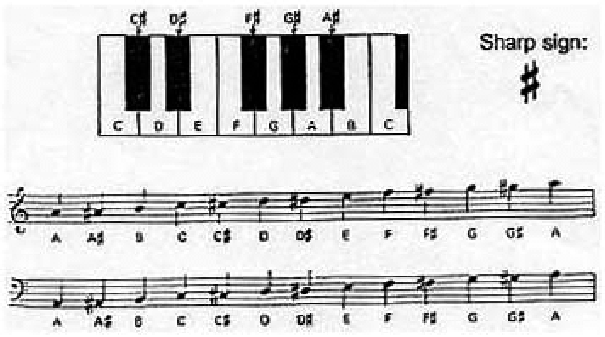
In this section we provide background for the non-musician and a basic method of shaping a sinusoid so as to match tonal output from real instruments.

2.1. Background for the Nonmusician

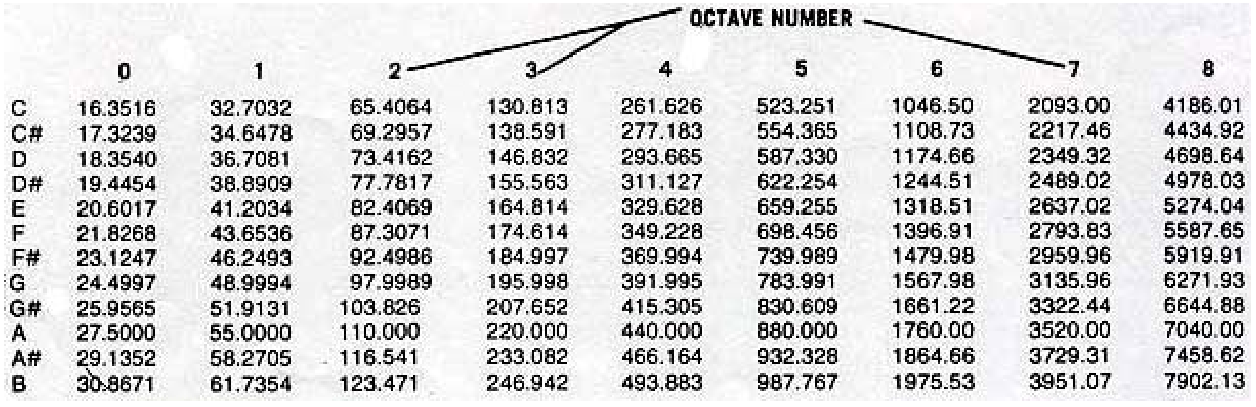
The fundamental element in music is the note or tone which is simply an oscillation; combinations of notes are called chords. On a piano each key defines a note and the frequencies of these notes are determined by their position relative to the 4th octave (middle) A which is set to 440Hz. Each octave up, down produces a note with twice, half the frequency respectively. With 12 notes per octave and the doubling relation, we can easily see that the frequency ratio of any two consecutive notes is $\sqrt[12]{2}$. Frequency, duration, and loudness data for producing music is efficiently coded as in Figure 1.



(a)


C♯ D♯ F♯ G♯ A♯
Sharp sign:
♯
C D E F G A B C
A A♯ B C C♯ D D♯ E F F♯ G G♯ A
A A♯ B C C♯ D D♯ E F F♯ G G♯ A

(b)



	OCTAVE NUMBER								
	0	1	2	3	4	5	6	7	8
C	16.3516	32.7032	65.4064	130.813	261.626	523.251	1046.50	2093.00	4186.01
C#	17.3239	34.6478	69.2957	138.591	277.183	554.365	1108.73	2217.46	4434.92
D	18.3540	36.7081	73.4162	146.832	293.665	587.330	1174.66	2349.32	4698.64
D#	19.4454	38.8909	77.7817	155.563	311.127	622.254	1244.51	2489.02	4978.03
E	20.6017	41.2034	82.4069	164.814	329.628	659.255	1318.51	2637.02	5274.04
F	21.8268	43.6536	87.3071	174.614	349.228	698.456	1396.91	2793.83	5587.65
F#	23.1247	46.2493	92.4986	184.997	369.994	739.989	1479.98	2959.96	5919.91
G	24.4997	48.9994	97.9989	195.998	391.995	783.991	1567.98	3135.96	6271.93
G#	25.9565	51.9131	103.826	207.652	415.305	830.609	1661.22	3322.44	6644.88
A	27.5000	55.0000	110.000	220.000	440.000	880.000	1760.00	3520.00	7040.00
A#	29.1352	58.2705	116.541	233.082	466.164	932.328	1864.66	3729.31	7458.62
B	30.8671	61.7354	123.471	246.942	493.883	987.767	1975.53	3951.07	7902.13

(c)

Figure 1: (a) Octave codes, (b) Note codes, (c) Frequency data.

2.2. Amplitude Modulation of Tones

The sound output of musical instruments does not immediately build up to its full intensity nor does the sound fall to zero intensity instantaneously. It takes a certain amount of time for the sound to build up in intensity and a certain amount of time for the sound to die away. The period of time during which a musical tone is building up to some amplitude (volume) is called the “attack time” and the time required for the tone’s intensity to partially die away is called its “decay time.” The time for final attenuation is called the “release time.” Many instruments allow the user to hold the tone for a period of time which is known as the “sustain time” so that various note durations can be achieved. The amplitude of the tone can “fit” inside a curve often called the Attack-Decay-Sustain-Release (ADSR) envelope as depicted in Figure 2. Typical ADSR envelopes are given for the guitar and piano [Figures 3a and 3b].

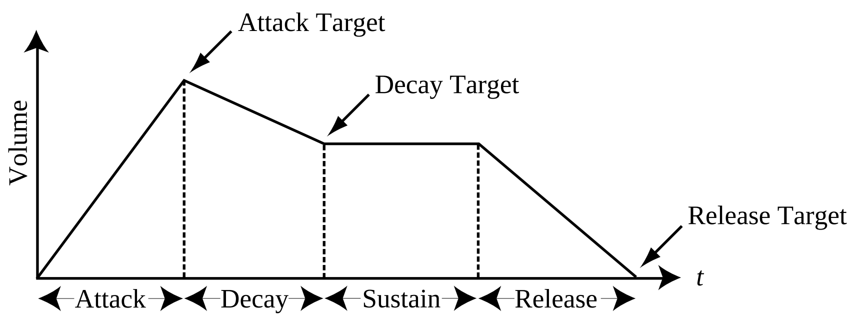


Figure 2: Classic ADSR envelope

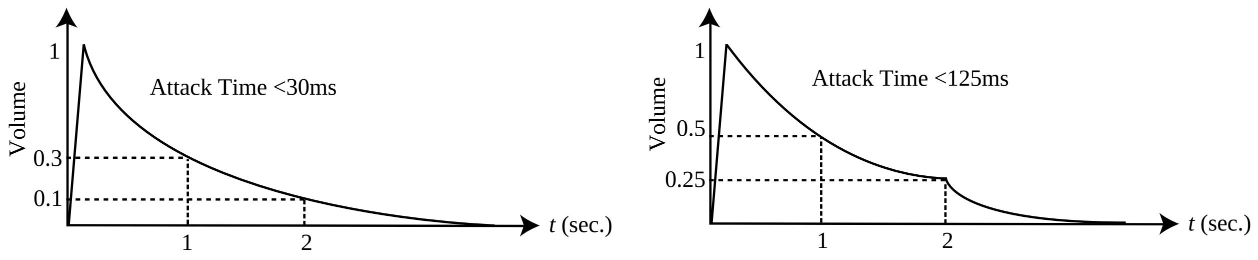


Figure 3: (a) ADSR envelopes for guitar, (b) ADSR envelopes for piano.

A synthesizer duplicates the intensity (volume) variation of the tone by multiplying (modulating) the amplitude of the sinusoid with a scale factor dictated by the ADSR envelope, $a(t)$

$$y(t) = a(t) \times x(t). \tag{1}$$

The resulting signal, $y(t)$ is referred to as the amplitude-modulated (AM) tone and the process is illustrated in Figure 4. Other methods of shaping basic tones have been investigated and include the more popular (but more complicated) frequency-modulation techniques [1].

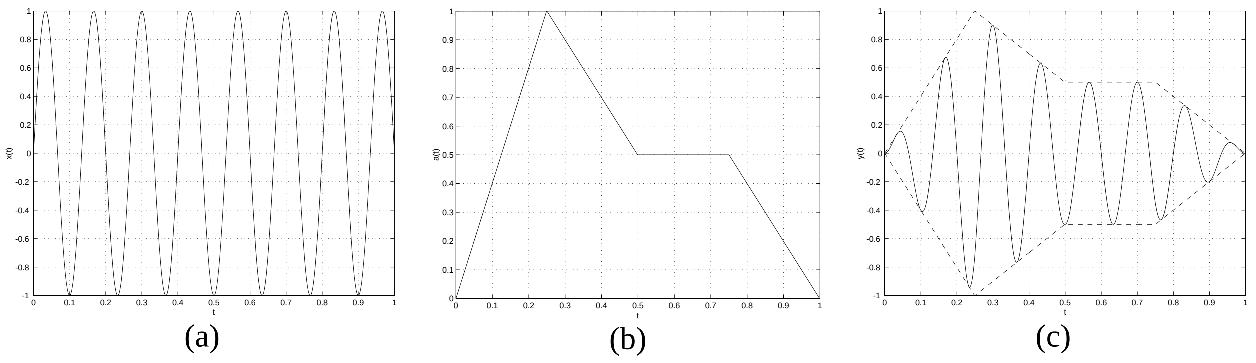


Figure 4: (a) Sinusoid, (b) ADSR envelope, (c) Amplitude Modulated (AM) sinusoid.

3. Envelope Model and Implementation

In this section we develop the MATLAB tools used to amplitude modulate a tone.

3.1. MATLAB Implementation

The implementation of a music synthesizer (AM-based) involves three codes: 1) tone synthesizer or sinusoid generator, 2) ADSR envelope generator, 3) composer/player code. We assume digital synthesis at a rate of f_s = 16,000 samples per second. At this rate we are able to reproduce all piano frequencies given in Figure 1c according to Nyquist theory [2].

3.2. Sinusoid Generator

A continuous-time sinusoid can be expressed as

$$x(t) = \sin(2\pi f t) \tag{2}$$

where *f* is the frequency of the sinusoid. Samples of (2) which are taken *T* seconds apart (called the sample period) can be expressed as

$$\begin{aligned} x(n) &= \sin(2\pi f n T) \\ &= \sin(2\pi n f / f_s) \end{aligned} \tag{3}$$

where *n* is the sample index and $f_s = 1 / T$ is the sample rate. Here we assume the sinusoid has unit amplitude and zero phase angle. The unit amplitude is chosen to avoid playback distortion since the sound card clips all sample magnitudes greater than one. Equation (3) then provides the formula for sinusoid generation and Figure 5 illustrates a MATLAB function which returns a vector of synthesized sinusoid samples.

```
function x = singen(f,fs,duration)
% Input
%   f        - frequency of sinusoid
%   fs       - sampling frequency
%   duration - duration of signal in seconds
% Output
%   x        - vector of sinusoid samples

n = [0:fs*duration]';
x = sin(2*pi*f/fs*n);
```

Figure 5: MATLAB implementation of sinusoid generator

3.3. Envelope Generator

As described earlier, the envelope will give the sinusoid a volume characteristic which as a first approximation, imitates that of a real instrument. The envelope values are stored as a single vector so that a simple element-by-element product between the sinusoid vector and the envelope vector yields the amplitude modulated sinusoid. The envelope is constructed one segment (A, D, S, and R) at a time. We approximate each segment with a simple exponential which rises or decays asympotically to the target value as in Figure 2. This approximation then leads to a simple digital filter implementation (difference equation) which students are familiar with, whose response yields samples of an exponential curve. In addition, we allow for a gain parameter to control the speed at which the exponential reaches the target value. The difference equation is given by a single-pole filter

$$a(n) = \hat{a}g + (1-g)a(n-1) \tag{4}$$

where $a(n)$ are the envelope values, $\hat{a}$ is the target value, and g is the gain parameter.

Figure 6 illustrates a MATLAB function which returns a vector of ADSR envelope values. For simplicity, we exclude a decay segment and assume an envelope duration of 1 second or 16,000 samples or a whole note. Most students eventually recode the ADSR routine so that the sustain segment varies with the duration of the note since in reality, music is composed of many durations of notes not just whole notes. In this case one simply makes the modification

sustain_duration = note_duration – attack_duration – release_duration. (5)

An example envelope (similar in nature to a piano) can be generated with the MATLAB calls in Figure 7.

3.4. Composer/Player Code

The final code segment generates sinusoids with the proper frequency and an ADSR envelope to amplitude modulate the sinusoid. While there is great flexibility here, a simple MATLAB code is given in Figure 8 (assuming the ADSR vector from Figure 7). In order to play a full song, the modulated sinusoid vectors would be concatenated together before playback. Finally, chords can be built by adding individual notes together and likewise, the chords would be concatenated before playback. Note that the addition of notes can yield sample values which may have magnitude greater than one and as noted in Section 3.2 there will be playback distortion. The simple fix is given in Figure 9.

```
function a = adsr_gen(target,gain,duration)
% Input
%   target   - vector of attack, sustain, release target values
%   gain     - vector of attack, sustain, release gain values
%   duration - vector of attack, sustain, release durations in ms
% Output
%   a        - vector of adsr envelope values
fs = 16000;
a = zeros(fs,1);  % assume 1 second duration ADSR envelope
duration = round(duration./1000.*fs); % envelope duration in samp

% Attack phase
start = 2;
stop = duration(1);
for n = [start:stop]
   a(n) = target(1)*gain(1) + (1.0 - gain(1))*a(n-1);
end

% Sustain phase
start = stop + 1;
stop = start + duration(2);
for n = [start:stop]
   a(n) = target(2)*gain(2) + (1.0 - gain(2))*a(n-1);
end

% Release phase
start = stop + 1;
stop = fs;
for n = [start:stop]
   a(n) = target(3)*gain(3) + (1.0 - gain(3))*a(n-1);
end;
```

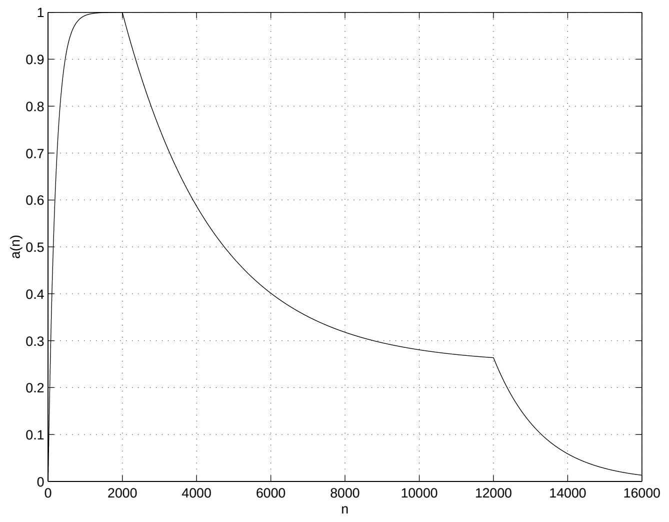
Figure 6: MATLAB implementation of attack, decay, sustain, release generator.

4. Example

The exercise in the undergraduate DSP course was to construct an “interesting” envelope and use the MATLAB synthesis tools to synthesize a song. At a minimum, students simply played a scale (amplitude modulated sinusoids over one octave) while others modified the ADSR_GEN tool for variable sustain (variable note durations) and included options for chords. One very impressive piece was a 25 second Bach musette which has been posted to the author’s web page in the form of a WAV file [3].

```
»target = [0.99999;0.25;0];
»gain = [0.005;0.0004;0.00075];
»duration = [125;625;250];
»adsr = adsr_gen(target,gain,duration);
»plot(adsr);
```

(a)



(b)

Figure 7: (a) MATLAB code for sample ADSR envelope, (b) Sample ADSR envelope.

```
»f0 = 440;fs = 16000;
»x = singen(f0,fs,1-1/fs);
»y = a .* x;  % Modulate
»sound(y,fs);
```

Figure 8: MATLAB code segment to generate sinusoid and modulate with adsr vector.

```
»y = y ./ max(abs(y)); % Normalize samples
```

Figure 9: Sample normalization for accurate reproduction.

5. Conclusions

In this paper we have developed a simple exercise in computer music for a typical undergraduate course in DSP. The exercise consists of three MATLAB codes which synthesizes a tone (sinusoid), generates an ADSR envelope used to amplitude modulated the tone, and builds a song from the modulated tones. A sample song is provided (through Internet download) which illustrates the power of the technique. While the target application is computer music, the ideas can be extended to more general ideas in waveform synthesis.

Bibliography
[1] J. Chowning, "The synthesis of complex audio spectra by means of frequency modulation," *Journal of the Audio Engineering Society*, Sept. 1973, vol. 21, no. 7, pp.526–534.
[2] S. Orphanidis, *Introduction to Signal Processing*, Prentice-Hall, Upper Saddle, NJ., 1996.
[3] A. Roman, "Bach Musette using MATLAB Tools," available at `http://telsat.nmsu.edu/~pdeleon/Research/Publications/Bach_Musett.wav`, Feb. 2000.

PHILLIP DE LEON
Phillip De Leon is an Assistant Professor in the Klipsch School of Electrical and Computer Engineering at New Mexico State University. In addition, he is Associate Director for the Center for Space Telemetering and Telecommunications. Dr. De Leon received his B. S. Electrical Engineering and B. A. Mathematics from the University of Texas at Austin in 1989 and 1990 respectively. In 1990, he was awarded the AT&T Bell Laboratories' Cooperative Research Fellowship Program (CRFP) fellowship for graudate studies. He received his M. S. and Ph.D. in Electrical Engineering in 1992 and 1995 respectively.