Wavetable Synthesis

In this lecture we will examine a popular method to synthesize waves using a lookup table.

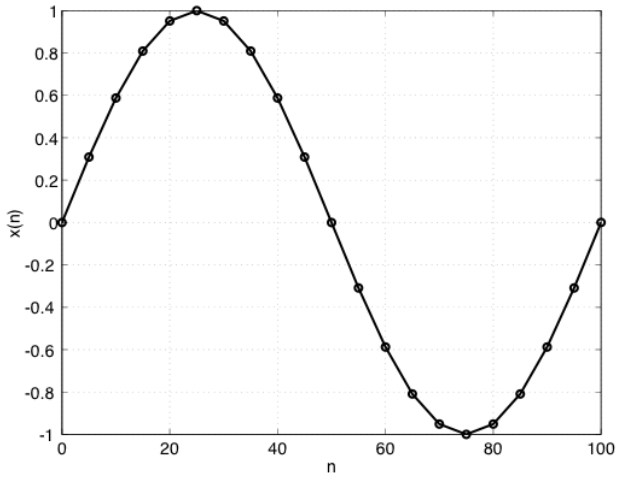
Sine Wave Synthesis

The generation of sinusoids or tones or waves is a widely used function of DSPs in audio, communications, and control applications. High-speed, high-precision DSPs are capable of synthesizing stable and low distortion sinusoids of any frequency. These sinusoids can be produced digitally using either truncated Taylor series or "Lookup Tables." We will examine the later.

Suppose we calculate the values of L evenly-spaced points on a sinusoid

$$x[n] = \sin(2\pi n / L), \quad 0 \le n \le L-1$$

and store them in memory to form a lookup table.



Base Address	0
Base Address + 1	$\sin(2\pi/L)$
Base Address + 2	$\sin(4\pi/L)$
	.
	.
	.
Base Address + L − 1	$\sin[2\pi(L-1)/L]$

If we read out one table entry every sample period, the frequency of the sinusoid produced at the D/A is then

$$\omega_0 = 2\pi / L \quad \text{(rads/sample)}$$

or

$$\omega_0 = \frac{2\pi}{L} f_s \quad \text{(rads/s)}$$

or

$$f_{\text{FUND}} = \frac{f_s}{L} \quad \text{(Hz)}$$

where f_s is the sample rate of the D/A. When one table entry is read out every sample period, the frequency of the sinusoid is referred to as the *fundamental table frequency* (FTF) or f_{FUND}. The *fundamental period*, T_{FUND} of the sinusoid is given by

$$T_{\text{FUND}} = 1/ f_{\text{FUND}}.$$

Example: Assume the sampling rate is f_s. = 16,000 samples/second and the lookup table has L = 256 entries. Then

$$\begin{aligned} f_{\text{FUND}} &= 16000/256 \\ &= 62.5 Hz \end{aligned}$$

and

$$T_{\text{FUND}} = 256/16,000 = 16 \text{ ms}.$$

Other frequencies can be generated from the lookup table through the use of a *table increment*. If the table increment, Δ = 1, the table entries are read consecutively; if Δ = 2, every other table entry is read; if Δ = 3, every third table entry is read; and so on. Using a table increment, we cycle through the lookup table Δ-times faster and thus the frequency of the sinudoid at the D/A will be

$$f = \Delta f_{\text{FUND}},\ 1 \le \Delta \le L/2.$$

The maximum value Δ can be is L / 2 since at least two samples per cycle are required to synthesize a sine wave without aliasing (result of Nyquist theory). The total harmonic distortion (THD) of the synthesized wave depends on the length of the table, L and the accuracy (number of bits of precision) of the data stored in the lookup table. Clearly, if Δ is an integer, only those frequencies which are integer multiples of the fundamental frequency can be generated

Δ	f (f_s = 16kHz, L = 256)
1	62.5
2	125.0
.	.
.	.
.	.
L / 2	8,000.0

Wavetable Synthesis for Fixed-Point DSP (TI 6416)

Generating the Lookup Table

The first step in synthesizing sinusoids is store the sinusoid values in a lookup table or array as follows

$$\text{waveTable}[l] = \sin(2\pi l / L),\ 0 \le l \le L - 1$$

where L is the table length. Prof. De Leon has written a MATLAB program, WaveTableMaker.m, that will generate the table values. These values can then be pasted into the initialize_program.c file (see below).

Sinusoid Synthesis with Integer Delta

Synthesis with an integer delta is straightforward. We use an offset into the lookup table to get the value and increment the offset by delta each sample period. The offset is always taken modulo L (table length) so that we do not address beyond the array. The following lines of code in the appropriate files will implement wavetable synthesis with an integer table increment or delta.

user_data.h
```
#ifndef USER_DATA
#define USER_DATA

void initialize_program();
void process_signal(short inputRight, short inputLeft, short *outputRight,
short *outputLeft);

#define TABLELENGTH 256

extern const short waveTable[];
extern short delta;     /* lookup table increment */
extern short offset;    /* accumulated delta */

#include "util.h"

#endif
```

initialize_program.c
```
#include "user_data.h"

/*************************************/
/* Global variable initializations here */
/*************************************/
const short waveTable[TABLELENGTH] = {
   0, 804, 1608, 2411, 3212, 4011, 4808, 5602, 6393, 7180,
...
   -4808, -4011, -3212, -2411, -1608, -804};

short delta = 2;  /* lookup table increment */
short offset = 0; /* accumulated delta */

void initialize_program()
{
}
```

lookup_waveIntDelta.c
```
#include "util.h"

short lookup_waveIntDelta(short L, const short *table, short delta, short
*Offset)
{
      short y;

      y = table[*Offset];     /* get table entry */
      *Offset += delta;       /* accumulate delta */
      *Offset = *Offset % L;  /* Offset always mod L */
      return y;
}
```

process_signal.c
```
#include "user_data.h"

void process_signal(short inputRight, short inputLeft, short *outputRight,
short *outputLeft)
{
      /*******************************/
      /* Process right channel sample */
      /*******************************/
      *outputRight = lookup_waveIntDelta(TABLELENGTH, waveTable, delta,
&offset);

      /*******************************/
      /* Process left channel sample */
      /*******************************/
      *outputLeft = inputLeft;
}
```

If Δ is restricted to an integer, the sinusoid generator is limited to integer multiples of the FTF. To generate arbitrary frequencies, we must allow for the possibility of a real-valued Δ, that is Δ composed of an integer and fractional part. When fractional values of Δ are used, required table values between table entries must be estimated. In the next section, we describe a linear-interpolation method for table entry estimation.

Sinusoid Synthesis with Real Delta: Linear Interpolation Method

In order to synthesize a sinusoid of any frequency with low distortion, an interpolation method must be used together with table lookup. By using interpolation, sinusoid values between table entries can be represented more accurately as in figure (a) below. The simplest interpolation method is linear interpolation

$$y = mx + b$$

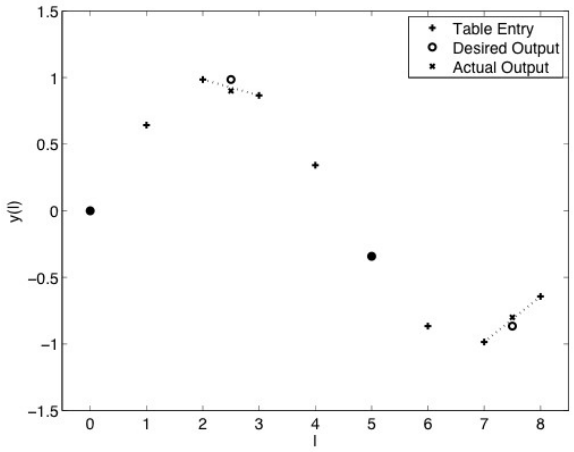
where

m = waveTable[l + 1] − waveTable[l]
b = waveTable[l]
x = fractional part of offset with 0 < x < 1.0
y = approximated sample, waveTable[$l + x$]

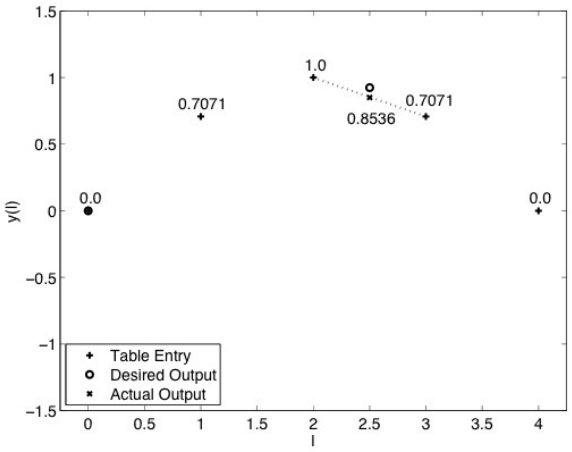
and waveTable[l] denotes the lth value in the wavetable.

Example: Let the parameters for the sinusoid synthesis be L = 8, Δ = 2.5, and l = 2. We then have the samples illustrated in figure (b) below in memory. In this case we have

m = 0.7071 – 1.0 = –0.2929
b = 1.0
x = 0.5
$y = mx + b$ = (–0.2929)(0.5) + 1.0 = 0.8536



(a)



(b)

Algorithm for Linear Interpolation Method

The basic idea is to accumulate the real-valued delta as an offset into the table. The integer part of the offset will point us to the first of the two successive values (*l*) while the fractional part of the offset will determine the distance between the successive values (*x*). Since the DSP is fixed-point, the integer parts of delta and offset will be stored as the upper 16 bits in an int while the *unsigned* fractional part of the delta and offset will be stored as the lower 16 bits of the int. The accumulation of delta into the offset using fixed-point is then

intOffset.UfracOffset = intOffset.UfracOffset + intDelta.UfracDelta

CALCULATE_SLOPE
Let intOffset denote the integer part of the offset and x1, x2 denote waveTable[intOffset], waveTable[intOffset+1] respectively. x1 and x2 are the two successive table values between which we must interpolate. The slope of the line across the two points is simply the "rise over the run" where the "rise" is x2 – x1 and the "run" is one.

CALCULATE_INTERPOLATED_VALUE
Let fracOffset denote the *signed* fractional part of the offset. The interpolated value, y is simply
(x2 – x1)*fracOffset + x1.

UPDATE_OFFSET
The new offset is incremented by delta as above and intOffset is taken modulo *L*.

Code for Linear Interpolation Method

The following lines of code in the appropriate files will implement wavetable synthesis with an integer table increment or delta.

user_data.h

```
#ifndef USER_DATA
#define USER_DATA

void initialize_program();
void process_signal(short inputRight, short inputLeft, short *outputRight,
short *outputLeft);

#define TABLELENGTH 256

extern const short waveTable[];
extern int delta;                    /* lookup table increment */
extern short intOffset;              /* integer part of accumulated delta */
extern unsigned short UfracOffset;  /* unsigned frac part of accumu delta */

#include "util.h"

#endif
```

initialize_program.c

```
#include "user_data.h"

/*************************************/
/* Global variable initializations here */
/*************************************/
const short waveTable[TABLELENGTH] = {
   0, 804, 1608, 2411, 3212, 4011, 4808, 5602, 6393, 7180,
...
   -4808, -4011, -3212, -2411, -1608, -804};

int delta;
short intOffset = 0;              /* integer part of accumulated delta */
unsigned short UfracOffset = 0;   /* fractional part of accumulated delta */

void initialize_program()
{
/* put integer, unsigned fractional part of delta in upper, lower 16 bits */
      delta = (int)(2<<16) | (int)((16384 <<1));  /* delta = 2.5 */
}
```

lookup_wave.c

```
#include "util.h"

short lookup_wave(short L, const short *table, int delta, short *intOffset,
unsigned short *UfracOffset)
{
      short fracOffset, x1, x2;
      int offset, y;

      x1 = table[*intOffset];  /* get consecutive table entries */
      x2 = table[(*intOffset+1) % L];
      fracOffset = (*UfracOffset)>>1;  /* restore sign bit */
      y = (x2-x1)*(fracOffset) + (int)(x1<<15);  /* y = mx + b */

      /* build offset = intOffset.UfracOffset */
      offset = (((int)(*intOffset))<<16) | ((int)(*UfracOffset));
      offset += delta;  /* increment offset by delta */

      /* mask off fractional part */
      *UfracOffset = (short)(offset & 0x0000FFFF);

      /* mask off integer part */
      *intOffset = (short)((offset & 0xFFFF0000)>>16);
      *intOffset = *intOffset % L;  /* intOffset always mod L */

      return cround(y)>>15;   /* round output and back to 16 bits */
}
```

process_signal.c

```
#include "user_data.h"

void process_signal(short inputRight, short inputLeft, short *outputRight,
short *outputLeft)
{
      /*******************************/
      /* Process right channel sample */
      /*******************************/
    *outputRight = lookup_wave(TABLELENGTH, waveTable, delta, &intOffset,
&UfracOffset);

      /*******************************/
      /* Process left channel sample */
      /*******************************/
      *outputLeft = inputLeft;
}
```