

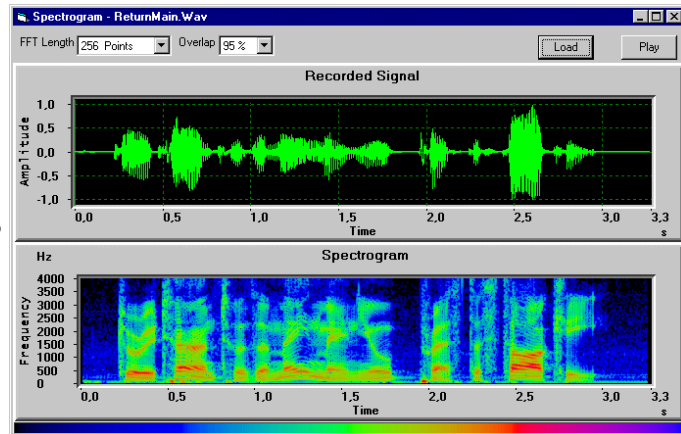
Spectrograph Output Utilizing a Circular Buffer

Produced by: Andrew Renton

Introduction

Spectrographs display the frequency information contained versus time as shown in picture 1 to the right. The relative magnitude of a given frequency is given by its color in the lower graph. The corresponding signal, an audio recording in this case, is displayed above.

Circular buffers essentially allow one to create a moving window to access a static array of information contiguously from any given starting point. Such buffers have many uses in a variety of programming applications including convolution algorithms and display outputs. In this case the circular buffer will facilitate a scrolling window so older data can be easily read in context.



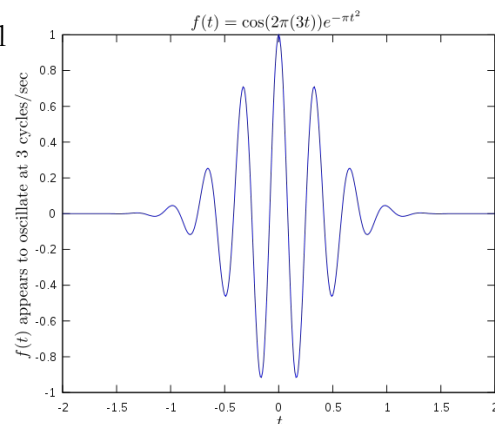
Picture: 1

This application note will describe a program to convert and display a signal into a spectrograph in real time. A general display method will be written with C based pseudo-code which should be easily applicable to most graphics programming interfaces.

Fast Fourier Transform

Frequency content representations of any signal requires the Fourier transform(FT). The FT integrates the full length of an input waveform, $f(x)$,and converts it into the complex value of that input at a given frequency as is shown in the equation below. This is repeated for a range of frequencies building a frequency representation of the input signal. The size of this range determines the accuracy with which the signal is modeled, with a larger range typically reducing the error with diminishing returns.

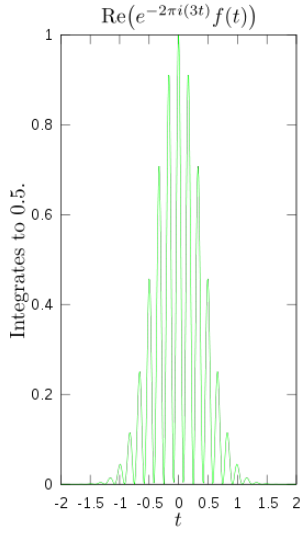
$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x \xi} dx,$$



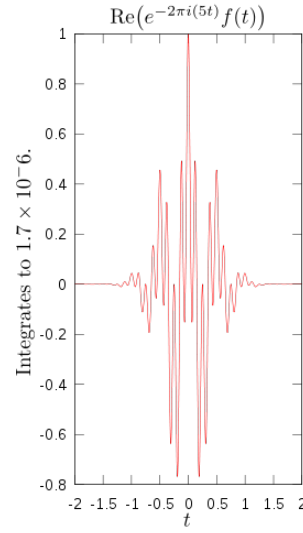
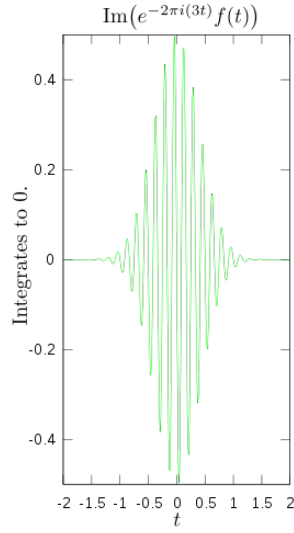
Picture: 2

Digital representation of signals fundamentally changes the method and form of the Fourier transform. Conceptually the process is the same, but continuous integration is not possible due to the discrete nature of samples in both range and time.

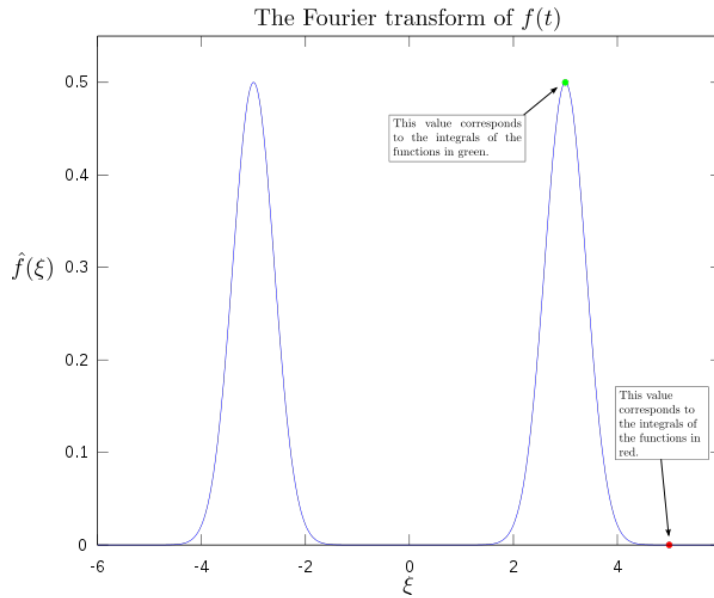
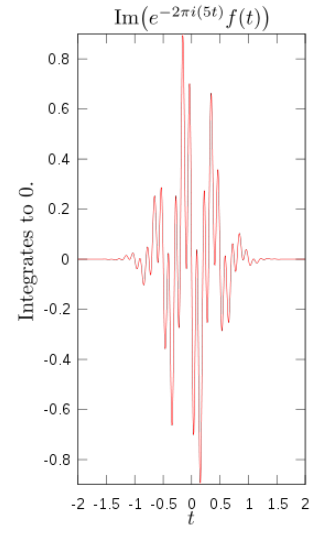
An example of the Fourier transform utilizing Matlab is illustrated in the pictures 2 through 5. Picture 2 shows the time domain signal to be transformed in this case a 3 Hz cosine damped by an exponential. Imaginary and real parts of the integrands for the frequencies 3 and 5 Hz are shown in pictures 3 and 4 respectively. The 3 Hz signal produces real and imaginary parts which directly correspond, leading to a magnitude greater than zero. At 5 Hz the real and imaginary parts, when taken as a magnitude, tends to zero. This process produces the values in the transform output shown in picture 5.



Picture: 3



Picture: 4



Picture: 5

Computational methods for like the one used for the Matlab output in picture 5 above utilize the fast Fourier transform (FFT). These functions take an array of values and recursively transforms the array into its frequency magnitude representation. Only half of the returned array will be useful, the second half is simply a mirrored copy of the first. This, and the recursive nature of the algorithm lead to better performance when array sizes corresponding to a power of two are used. Most algorithms will append zeros to arrays not conforming to this rule. Magnitudes are returned from ranges of frequencies corresponding to its index multiplied by the signals sampling rate divided by the FFT size.

$$index * samp_{rate} / FFT_N < frequency < (index + 1) * samp_{rate} / FFT_N$$

Libraries for the FFT have been written for nearly every platform which is capable of calculating it. As such I will not elaborate on the implementation of the FFT itself and will focus on the use of its output. Most FFT algorithms will have an interface which closely matches the one described above and used in examples to follow.

Circular Buffer

Computer programs typically store ordered data in the form of an array. To illustrate, a 5 slot array is shown below filled with the first 5 natural numbers. To access one of these numbers its index is used. In this case the index of 0 would correspond to the number 1. For many languages attempting to access a index out of the array's bounds will result in an error. Example 1, written in C, demonstrates iterating through an array as well as an erroneous loop which goes out of the array bounds. It is important to insure any use of an array falls within its bounds with appropriate error checking. Explanatory comment lines within the code are denoted by 3 asterisks(*).

Initial state of array

index	0	1	2	3	4
value	1	2	3	4	5

Table: 1

Imagine a number generator constantly producing the natural numbers which are to be read and output along with the previous four values. The program will use the 5 slot array above as its initial state. The writing process first fills the array(already shown)and then restarts at the beginning of the array. This is shown in example 2 assuming the pattern of natural numbers continues from the previous example. If the array is output in order as in example 1 the numbers will not be printed as an ascending list but in a haphazard order. To mitigate this the printing process must start iterating through

```
int arraysize = 5;
int array[arraysize] = {1,2,3,4,5};
int offset = 0, write_position=0;
while(1){ ***causes an endless loop
    *** reads and writes value to write_pos
    readvalue(write_position);
    ***write_pos = write_pos+1 = offset
    write_position++; offset = write_positon;
    for (n = 0; n<arraysize-1; n++){
        if(offset+n==arraysize)
            offset = -1*n;
        print (array[n+offset]);
    }
    ***This example would output 23456 then 34567 etc.
```

Example: 2

```
int arraysize = 5;
int array[arraysize] = {1,2,3,4,5};
***The loop below works
***and prints 12345
for (n = 0; n<arraysize-1; n++){
    print (array[n]);
}
***This loop goes out of bounds
*** after printing 12345
for (n = 0; n<arraysize; n++){
    print (array[n]);
}
```

Example: 1

the array at an arbitrary location in relation to the last written value and wrap around to finish printing the initial skipped values. In this case it would be best to offset the start of the loop one index after the last written index so that it's value is printed last. This causes a logic issue when the offset + the iterator n go out of the array bounds. This is remedied by testing for out of bounds values and if true setting their sum to zero (offset = -n). This restarts the iterator at the array head.

State of array after second read

index	0	1	2	3	4
value	6	7	3	4	5

Table: 2

Spectrograph Output

Spectrograph displays essentially combine the two concepts covered previously, FFTs and circular buffers, and will vary mostly in the graphics interface used to generate the output. I will use only one graphics oriented function to illustrate the concept `point()` or `pixel()` which would typically be defined as in the definition below. Code in example 3 utilizes the circular buffer concept to iterate through the array of FFT output arrays. Once an array is singled out it is printed as the array is in example 1. Instead of being output as a set of numbers the output represents the frequency components of a signal with lower frequencies being at the bottom

```
int offset = 0, write_position=0;
int Matrix [FFT_N/2] [Graph_Size]
while(1){ ***causes an endless loop
    *** reads and writes array to Matrix[n/a][write_pos]
    read_array(write_position);
    ***write_pos = write_pos+1 = offset
    write_position++; offset = write_positon;
    for (n = 0; n<Graph_Size; n++){
        if(offset+n==Graph_Size)
            offset = -1*n;
        for(x = 0; x<FFT_N/2; x++){
            point(x+n , y-x , Matrix[x][n])
        }
    }
}
```

Example: 3

Bottom left of pixel output in table form when offset = 0

...	...	...
p(x,y-3,Array[3][0])	p(x+1,y-3,Array[3][1])	...
p(x,y-2,Array[2][0])	p(x+1,y-2,Array[2][1])	...
p(x,y-1,Array[1][0])	p(x+1,y-1,Array[1][1])	...
p(x,y,Array[0][0])	p(x+1,y,Array[0][1])	...

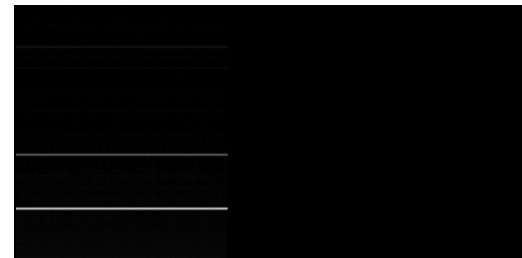
p(x,y,val) represents the point function above
each cell of the table represents a single pixel

Table: 3

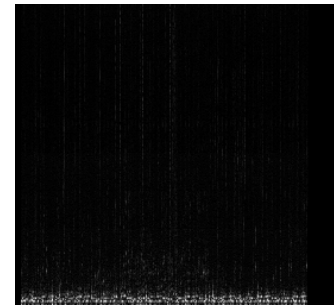
Point(x,y,color) :

x , y are the Cartesian coordinates of the pixel to change
color is a numerical representation of the color to use
I will use a range of 0 to 255 gray-scale values

Definition



Picture: 6



Picture: 7

Examples of output are shown above. Picture 6 is the response to a 1kHz sine wave. The thickest white line is the appropriate frequency position with the upper ones expressed due to harmonic noise. Picture 7 shows the response to an audio track. The large regular bars are indicative of percussive hits especially the snare.

Works Cited

[1] Kurz, Wilhelm. Spectrogram retrieved from <http://www.wilhelm-kurz-software.de/dynaplot/applicationnotes/spectrogram.htm>

[2] Proakis, John; Manolakis, Dimitris. Digital Signal Processing 4th edition (2007). Chapter 8

[3] Thenub314. (February 2009) Wikimedia Commons