Audio Noise Reduction in Windows Phone

This article shows an approach for audio noise reduction using Fast Fourier Transforms on Windows Phone.

09 Sep 2012

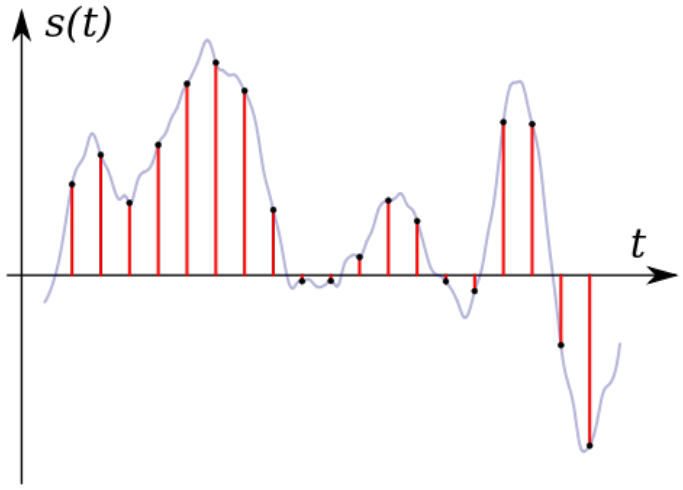
Fast Fourier Transform

This section provides a very simple and broad overview of the Fast Fourier Transform - the minimum needed to understand how the noise reduction algorithm works. For a slighly deeper view, see Sound pattern matching using Fast Fourier Transform in Windows Phone.

Fast Fourier Transform computes the DFT and transforms a function from the Time domain (physical signals) into another, which is called the frequency domain representation - in short a spectrum graph showing the frequencies present in the sample. The inverse Fast Fourier Transform does the reverse, transforming the frequency domain back into a physical signal.

The FFT requires an input function that is discrete. Such inputs are created by sampling a continuous function, such as a person's voice, a song or an ambient noise. The algorithm only applies to signals comprising a number of elements which is equal to 2^n and returns a set of complex numbers, the spectral components. The number of FFT elements is equal to the size of the time sample.

The second half of these complex numbers corresponds to negative frequencies and contains complex conjugates of the first half for the positive frequancies, and does not carry any new information.



How Noise Reduction Works

First we use Fourier analysis to find the spectrum of pure tones that make up the background noise. For each windowed sample of the background audio, we take a Fast Fourier Transform (FFT) and then statistics are tabulated for each frequency band - specifically the maximum level achieved by at least *n* sampling windows in a row, for various values of *n*. This spectrum is referred to as the "fingerprint" of the static background noise in the environment.

When recording, we then take the frequency spectrum of each short sample of audio and compare it to our fingerprint. Pure tones in the sample that aren't sufficiently louder than the fingerprint are probably noise, so we reduce their values in the spectrum (this general technique is called *spectral noise gating*).

We then compute the inverse FFT of the sample's "noise-reduced" spectrum to convert it back to a time domain, and play the audio. The result is the original sound, but with the frequencies associated with the background noise much reduced.

Similar techniques are used in high end noise-reduction headphones; the main difference being that these will often dynamically calculate the noise in "real time" using a second microphone.

Working with FFT in Windows Phone

- Download Dsp.zip and extract DSP.cs from the zip. Then add it into your project. **DSP.cs** provides a namespace called `DSP` and a class `FourierTransform` containing a set of funtions to compute the FFT.
- Refer to How to access and manage the Microphone raw data in WP7 for full instructions on how to manage the microphone on WP7.
- Here a very good Microphone sample code, the same used in this article.

Don't forget to include the namespace DSP

```
using DSP;
```

`Compute()`

Is the function in the namespace delegated to compute the FFT. Here the signature:

```
void Compute(UInt32 NumSamples, Double[] pRealIn, Double[] pImagIn, Double[] pRealOut, Double[] pImagOut, Boolean bInverseTransform)
```

- `NumSamples` Number of samples (must be power two)
- `pRealIn` Real raw data samples
- `pImagIn` Imaginary (optional, may be null), to be filled when calculating inverse Fourier Transform
- `pRealOut` Real coefficient output
- `pImagOut` Imaginary coefficient output
- `bInverseTransform` bInverseTransform when true, compute Inverse FFT

Cutting the frequencies

First create the array of double to store the noise fingerprint.

```
private double[] fingerprint;
```

We need a DispatcherTimer in order to manage the noise fingerprint detection. The detection time set is 4 sec, though one can set it to any value. It is observed that longer time durations (>10 secs)dosen't improve result, rather other background noises may interfere and the net resultant (fingerprint) may not be optimal.

```
DispatcherTimer dtFingerprint;

 // Timer to detect fingerprint
dtFingerprint = new DispatcherTimer();
dtFingerprint.Interval = TimeSpan.FromMilliseconds(4000);
dtFingerprint.Tick += new EventHandler(stopFingerprintDetection);

private void stopFingerprintDetection(object sender, EventArgs e)
{
            dtFingerprint.Stop();
            microphone.Stop();

            MessageBar.Text = "Noise fingerprint computed.";

            SetButtonStates(false, false, true);

            microphone.Start();

            UserHelp.Text = "Record";
            StatusImage.Source = microphoneImage;
}
```

`DispacherTimer` is included in `System.Windows.Threading namespace`.

Into my .xaml I added a checkbox component to enable/disable noise reduction during recording.

```
<CheckBox Content="Noise Reduction" Name="cb_noise_reduction" ... />
```

When the record button is pressed we allocate the fingerprint array and start the timer for detection.

```
private void recordButton_Click(object sender, EventArgs e)
{
            // Get audio data in 1/2 second chunks
            microphone.BufferDuration = TimeSpan.FromMilliseconds(100);

            // Allocate memory to hold the audio data
            buffer = new byte[microphone.GetSampleSizeInBytes(microphone.BufferDuration)];

            // Allocate memory to hold the audio data
            fingerprint = new double[ FFT.FourierTransform.NextPowerOfTwo((uint) microphone.GetSampleSizeInBytes(microphone.BufferDu

            // Set the stream back to zero in case there is already something in it
            stream.SetLength(0);

            WriteWavHeader(stream, microphone.SampleRate); // To save in .WAV format

            if ((bool)cb_noise_reduction.IsChecked)
            {
                dtFingerprint.Start(); // Start the noise finger print detection
            }
            else
            {
                SetButtonStates(false, false, true);
                UserHelp.Text = "Record";
                StatusImage.Source = microphoneImage;
            }

            // Start recording
            microphone.Start();

}
```

On `dtFingerprint` timeout recording begins. Inside the `Microphone.BufferReady` event handler.

```
private double cutoff = 0;

void microphone_BufferReady(object sender, EventArgs e)
 {
            // Retrieve audio data
            microphone.GetData(buffer);

            int index = 0;
```

```
            double[] sampleBuffer = new double[FFT.FourierTransform.NextPowerOfTwo((uint)buffer.Length)];

            for (int i = 0; i < buffer.Length; i += 2)
            {
                sampleBuffer[index] = Convert.ToDouble(BitConverter.ToInt16((byte[])buffer, i)); index++;
            }

            if (dtFingerprint.IsEnabled)
            {
                MessageBar.Text = "Computing noise fingerprint";

                double[] xre = new double[sampleBuffer.Length]; // Real part
                double[] xim = new double[sampleBuffer.Length]; // Immaginary part

                FFT.FourierTransform.Compute((uint)sampleBuffer.Length, sampleBuffer, null, xre, xim, false);

                double spectrum = 0;

                for (int i = 0; i < xre.Length; i++)
                {
                    spectrum = (float)(Math.Sqrt((xre[i] * xre[i]) + (xim[i] * xim[i]))); // Magnitude
                    if (spectrum > fingerprint[i])
                    {
                        fingerprint[i] = spectrum;
                    }
                }
            }
            else
            {
                MessageBar.Text = "Recording....";

                double cMagnitude = 0;
                // double cPhase     = 0;

                double[] xre = new double[sampleBuffer.Length]; // Real part
                double[] xim = new double[sampleBuffer.Length]; // Immaginary part

                double[] ixre = new double[sampleBuffer.Length]; // Real part
                double[] ixim = new double[sampleBuffer.Length]; // Immaginary part

                double[] fftoutput = new double[sampleBuffer.Length];
                byte[] output = new byte[buffer.Length];

                FFT.FourierTransform.Compute((uint)sampleBuffer.Length, sampleBuffer, null, xre, xim, false);

                for (int i = 0; i < xre.Length; i++)
                {
                    cMagnitude = (float)(Math.Sqrt((xre[i] * xre[i]) + (xim[i] * xim[i]))); // Magnitude

                    if (cMagnitude < (fingerprint[i] ))
                    {
                        xre[i] *= cutoff;  xre[(xre.Length - 1) - i] *= cutoff;
                        xim[i] *= cutoff;  xim[(xre.Length - 1) - i] *= cutoff;
                    }
                }

                FFT.FourierTransform.Compute((uint)xre.Length, xre, xim, ixre, ixim, true);

                index = 0;
                short tmp = 0;

                for (int i = 0; i < buffer.Length / 2; i++)
                {
                    tmp = (short)ixre[i];
                    output[index] = (byte)((short)tmp & 255); output[index + 1] = (byte)((((short)tmp) >> 8) & 255);
                    index += 2;
                }

                // Store the audio data in a stream
                //stream.Write(buffer, 0, buffer.Length);
                stream.Write(output, 0, output.Length);
            }

    }
```

Downloads

- DSP.zip has the full example code.

Summary

The article has shown an approach how to cut off some frequencies from your audio sample focused on Windows Phone. The theory is also valid for Qt/Symbian and S40 platforms.