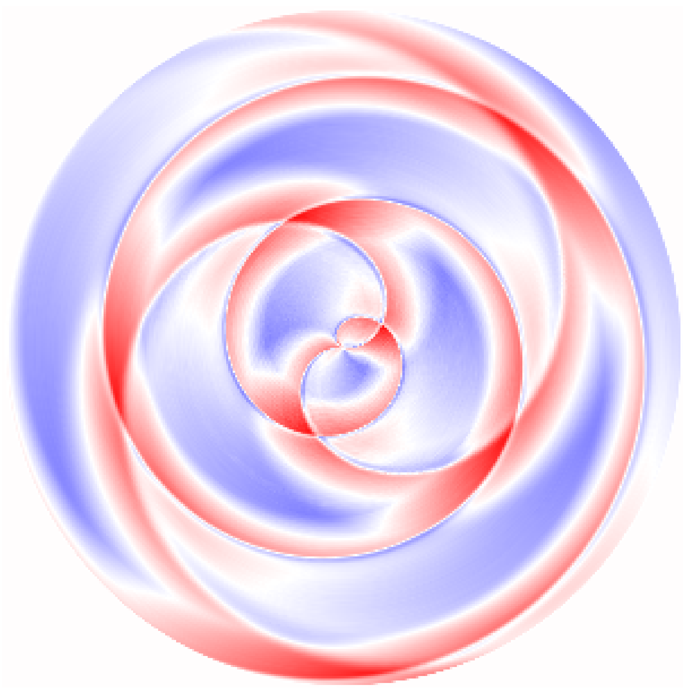
A time domain analysis method for pseudo-periodic signals

Fritz Menzer
fritz.menzer@epfl.ch

18. 3. 2001



Note: Each picture produced with the method described in this document is linked to the corresponding sound. To hear which sound produces the above picture, just click on it.

Contents

1 Introduction

There are many methods that analyse signals in time domain or frequency domain (using the Fourier transform). So why come up with some new method that works only with pseudo-periodic signals[1] and that doesn't give you any precise information (e.g. the intensities of the different harmonics)?

Even though my main motivation to publish this method is the beauty of the obtained pictures, there may be some practical usefulness in it. For example it can be rather difficult to say whether a sound was produced by an FM synthesizer or by a classical analog syntheziser. When analysed with the method described here FM sounds as well as analog sounds show caracteristic two-dimensional features.

This method is very similar to the method where a pseudo-periodic signal is divided in its periods and all the periods are displayed in a time-period graph (c.f. figure 1).

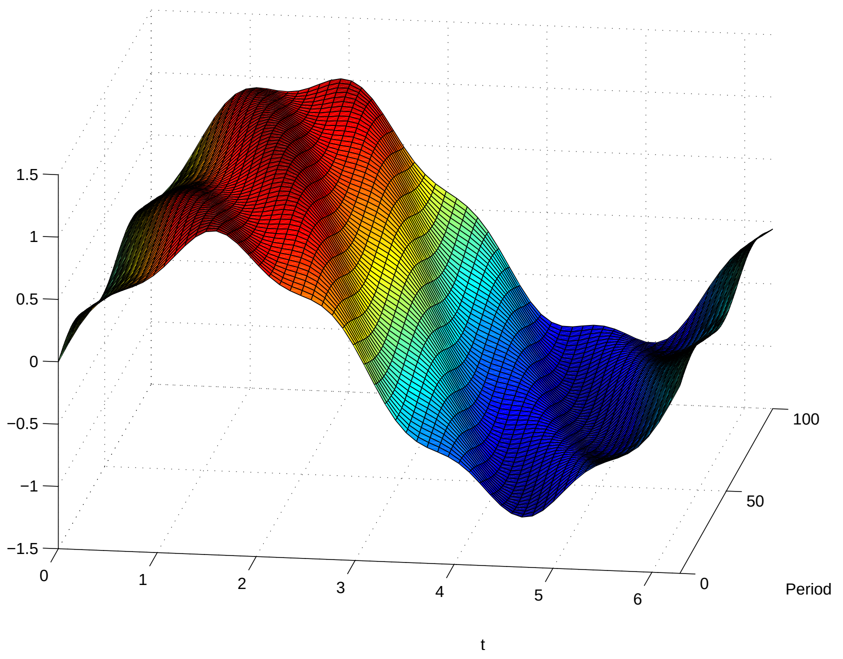


Figure 1: A time-period graph of the superposition of two detuned sine waves (frequency ratio ca. 1:5)

The method described here is basically the same as the time-period graph method, but in polar coordinates (where θ depends on the time and r depends mainly on the period). So it is interesting to compare those two methods.

The traditional time-period graph method has two problems:

- In discrete-time signals the period length is in general not an integer. So where should you cut?

[1]signals showing a period like periodic signals but differing slightly in their shape from one period to the next

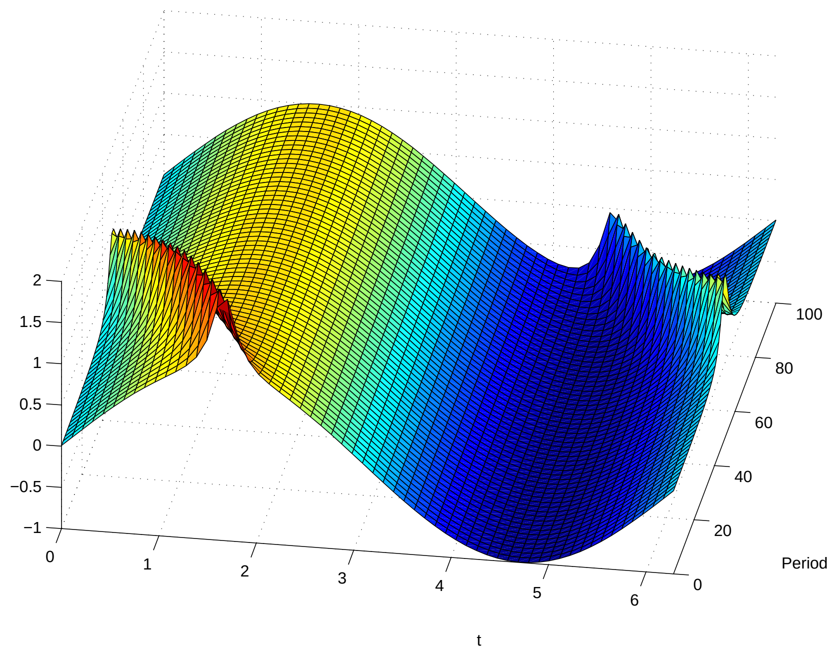


Figure 2: Here a "feature" (the peak) is moving over the period boundary.

- Where should you cut anyways? Most often there is no "canonical" division between periods. "Features" like maxima don't care about your division into periods and may move over period boundaries, meaning that they disappear on one side of your graph and appear on the other (c.f. figure 2).

The method described here solves those two problems, but creates new ones:

- A period is not displayed the same way if it is at the beginning of the signal as if it were at the end of the signal
- It is hard to do precise measurements in the graph obtained by using this method.

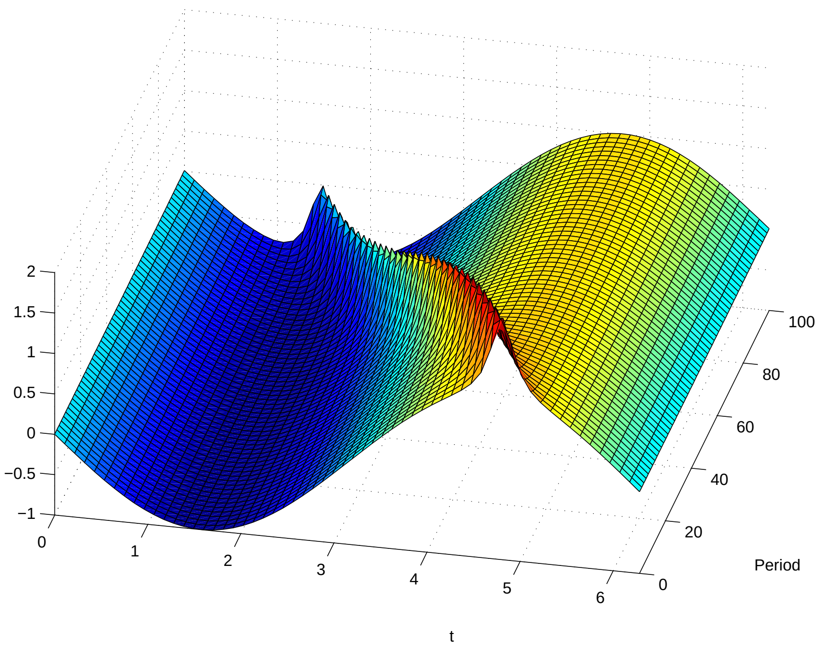


Figure 3: In order to get a better graph we have to move the period boundary.

2 Surface generation

2.1 Discrete-time case

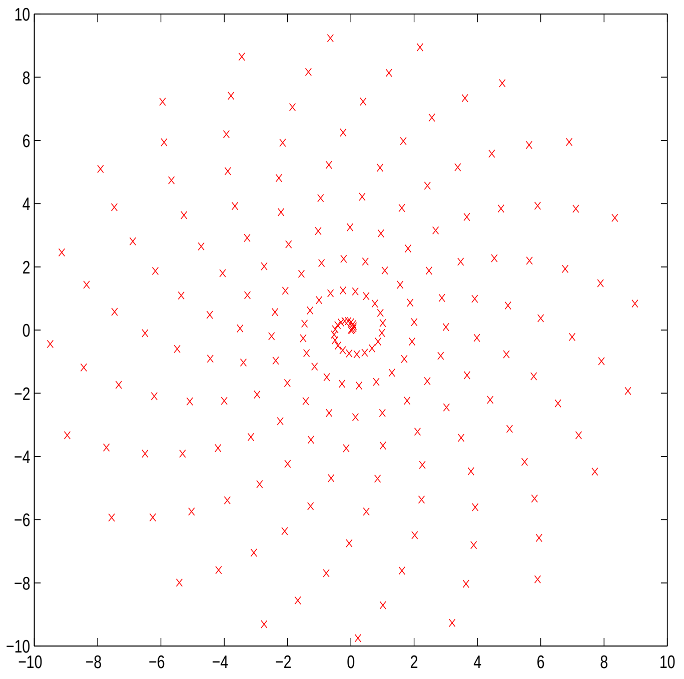


Figure 4: 200 points distributed according to eq. 1 with $T = 20.3$ and $\alpha' = 1$

Let $s[n]$ be a pseudo-periodic signal which has a period of $T \in I\!R$ samples. Let's assign to each sample $s[n]$ a point $(x[n], y[n]) \in I\!R^2$ using the following formula

$$(x, y)[n] = \frac{\alpha' n}{T} \left(\cos \frac{2\pi n}{T}, \sin \frac{2\pi n}{T} \right) \tag{1}$$

What is happening is basically that the signal get's "laid out" along a spiral in $I\!R^2$ (c.f. figures 4, 5).

If we consider (x[n],y[n],s[n]) being a series of points in $I\!R^3$ we can define a function $f(x, y)$ as the surface obtained by interpolating between those points. Of course $f(x, y)$ will depend on the interpolation algorithm chosen. The algorithm used to create the pictures in this article can be found in the appendix.

In the examples shown in this article the surface is displayed using a color map. Red means -1, white means 0 and blue $+1$. All samples have been normalized to $[-1, +1]$. In order not to introduce a DC component in the samples, they are centered around zero. So it may happen that only -1 or $+1$ is reached.

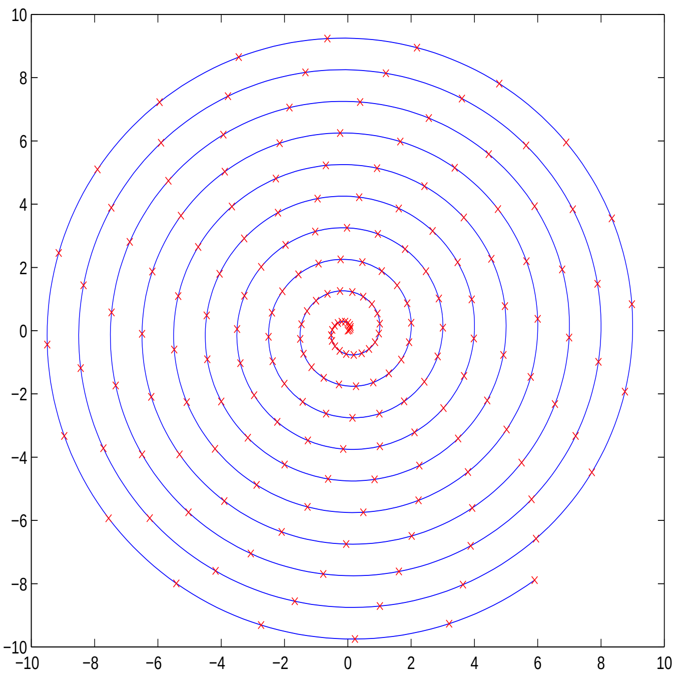


Figure 5: Same as figure 4 but with the spiral traced

2.2 Continuous-time case

There should be no problem to generalize the method shown in the previous section to a continuous-time signal $s(t)$. One method would be to sample $s(t)$ with a sampling interval t_s and let $t_s \to 0$.

But writing an interpolation algorithm for continuous-time signals would be even easier than for discrete-time signals since for each point only one interpolation needs to be done instead of three like in the code in the appendix.

3 Base frequency determination

From here on the text will refer to the base frequency of a signal rather than the period because in general one is more interested in the frequency, also because the frequency of a discrete-time signal is independent of the sampling frequency while the period (measured in samples) is not. Using the period T was just to show the problems that occur when T is not an integer. Of course it is equivalent to talk about the base frequency or the period, because one is determined by the other. In the continuous-time case their relation is $Tf = 1$ where $[T] = s$ and $[f] = Hz$. In discrete-time T is measured in samples and $Tf = f_s$ where f_s is the sampling frequency.

This section shows how to determine the base frequency empirically and what a certain pattern on the picture tells us about the base frequency of the sound.

The following pictures are all based on the same signal, but use different values of T to display it.

The example signal has a base frequency of 55Hz but we don't know that in advance so we use a trial-and-error method. Of course the base frequency could also be determined by using FFT, but it is amusing to see what happens when the frequency is not the correct value.

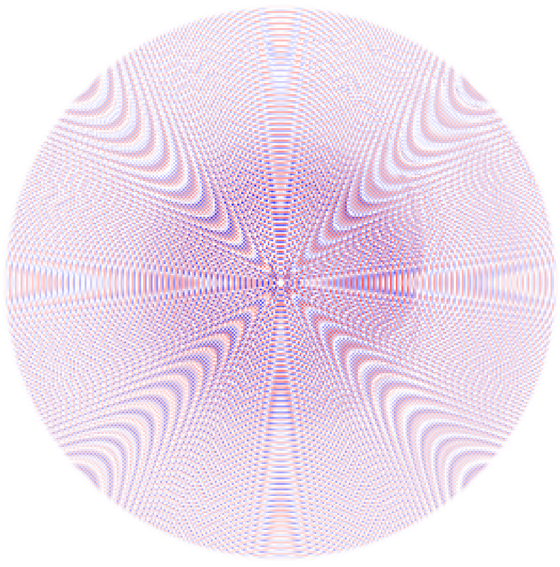


Figure 6: $f = 440Hz$; Completely off. Let's try a very low value to start with.

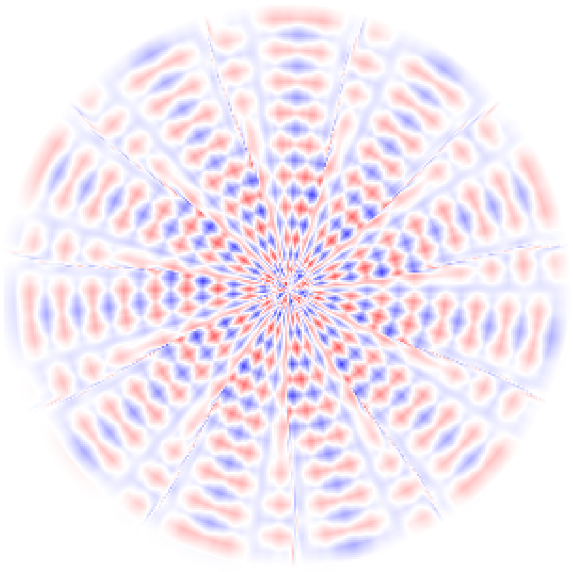


Figure 7: $f = 10Hz$; Ok. We seem to be at a sub-harmonic of the base frequency. Let's try the double frequency.

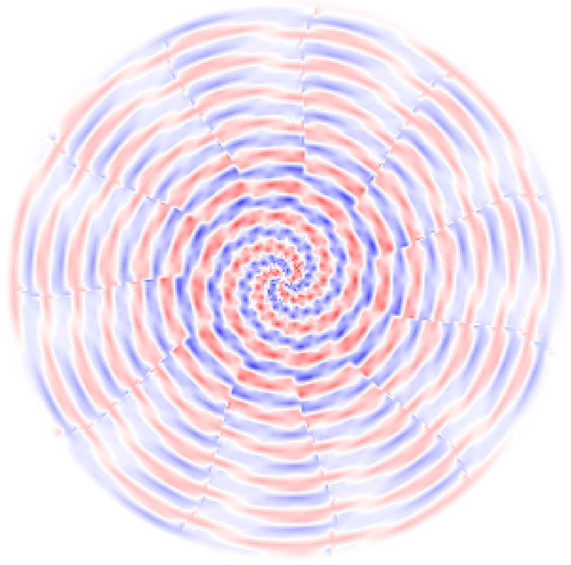


Figure 8: $f = 20Hz$; We can still see a pattern, but it seems to "turn" counter-clockwise. Let's try a lower frequency (would be a higher frequency if it turned clockwise).

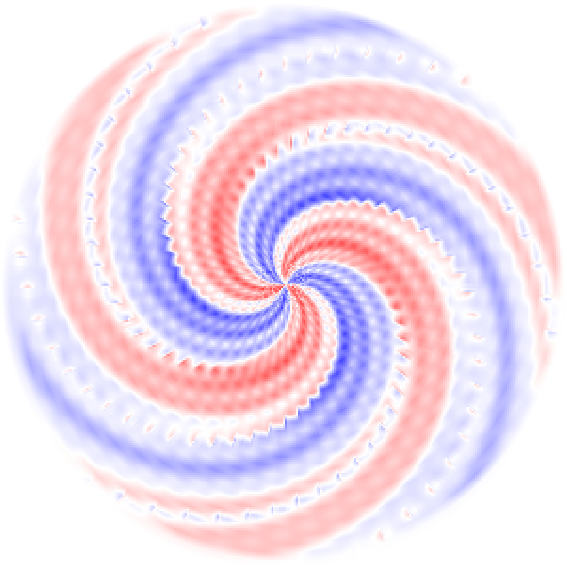


Figure 9: $f = 18Hz$; Better, but now it turns clockwise, Let's try a higher frequency.

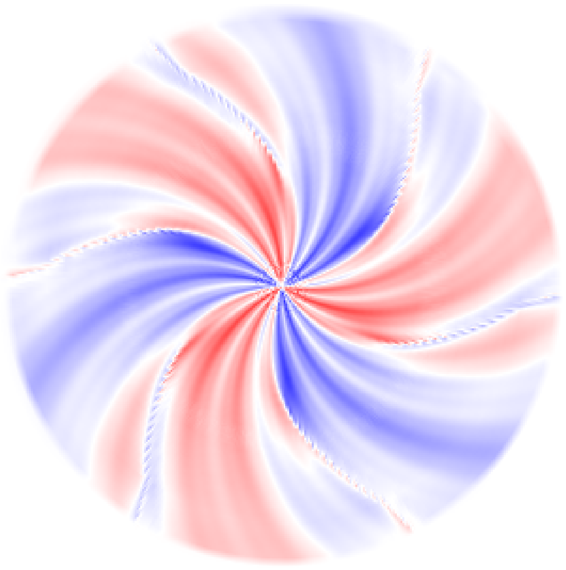


Figure 10: $f = 18.4Hz$; Quite good. But f should be a bit lower now.

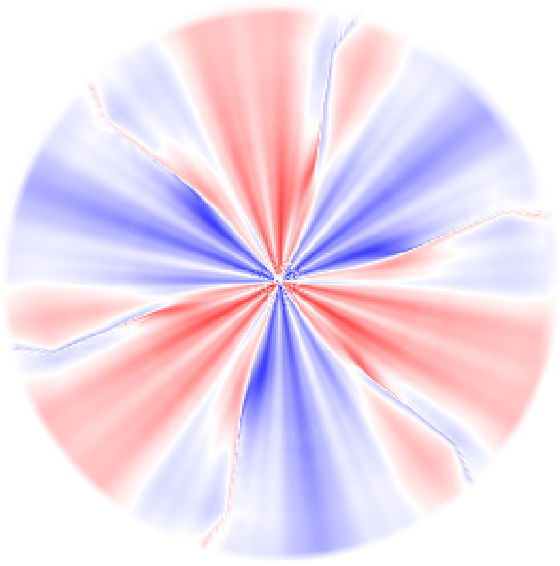


Figure 11: $f = 18.33Hz$; Great. But we see the same pattern repeated three times. This means that our frequency is three times lower than the sound's. So we use $55Hz$

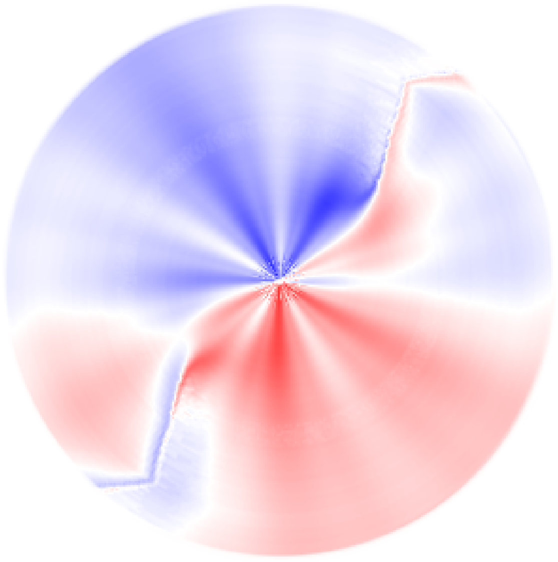


Figure 12: $f = 55Hz$; Finally we're there!

4 Examples

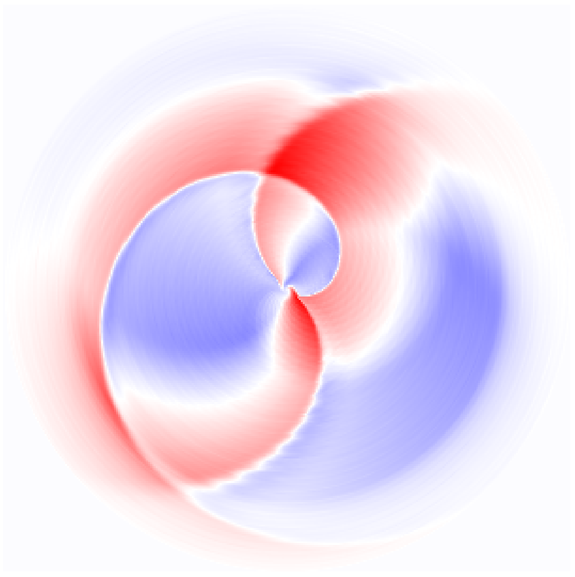


Figure 13: An analog synth brass sound (from Crumar Bit01)

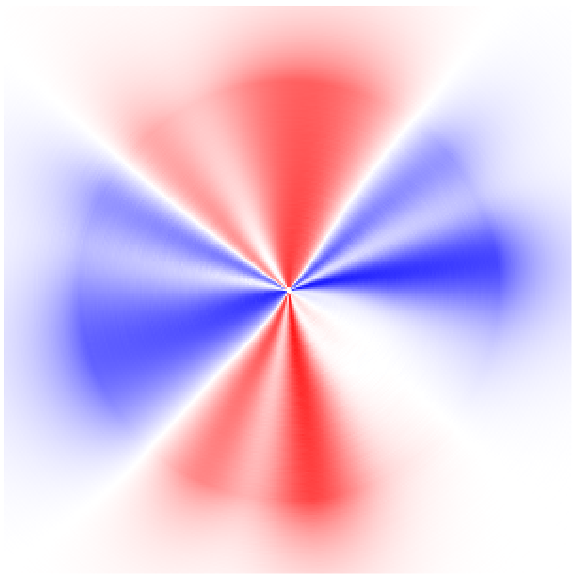


Figure 14: An FM bell sound (from Yamaha TX81Z). With a bit of phantasy you can see a bell in this picture. Thank you, Simona Cereghetti, for giving me this hint!

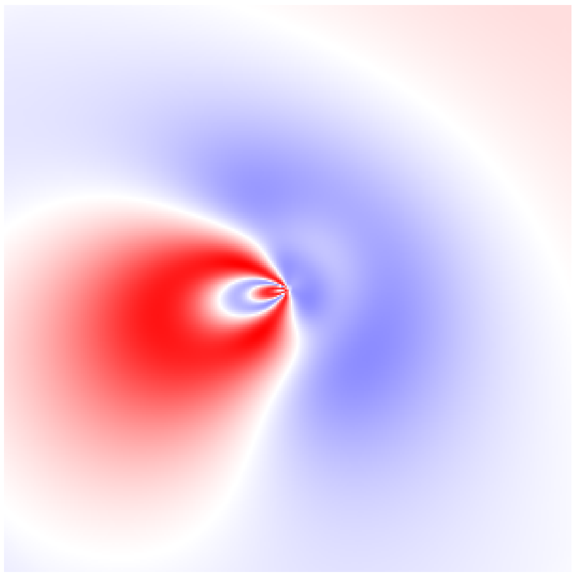


Figure 15: A bass sound produced with $\mathbb{R}^2$m

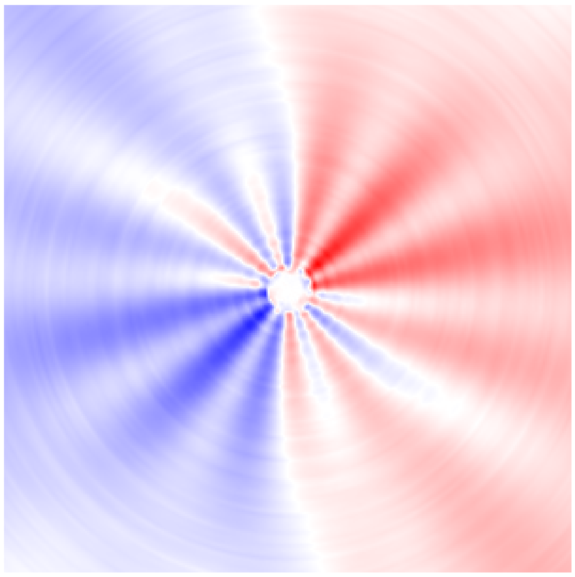


Figure 16: A guitar sound (steel string)

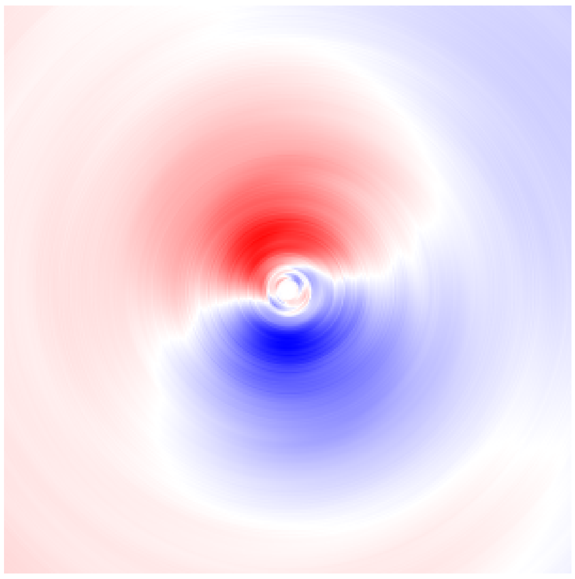


Figure 17: A piano sound at 435Hz

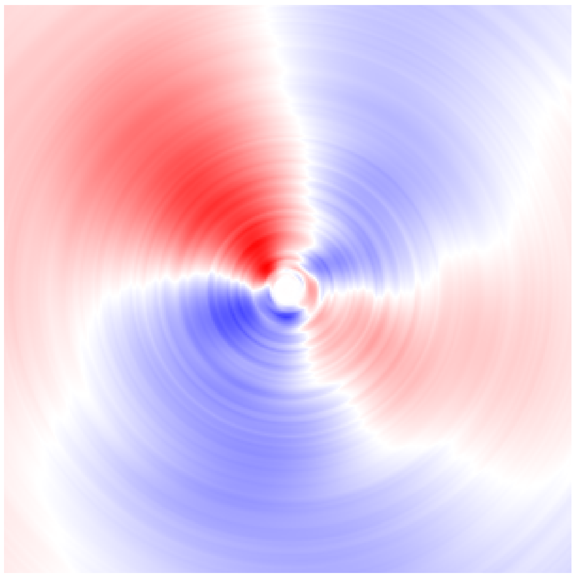


Figure 18: A sound from the same piano, one octave lower than in figure 17

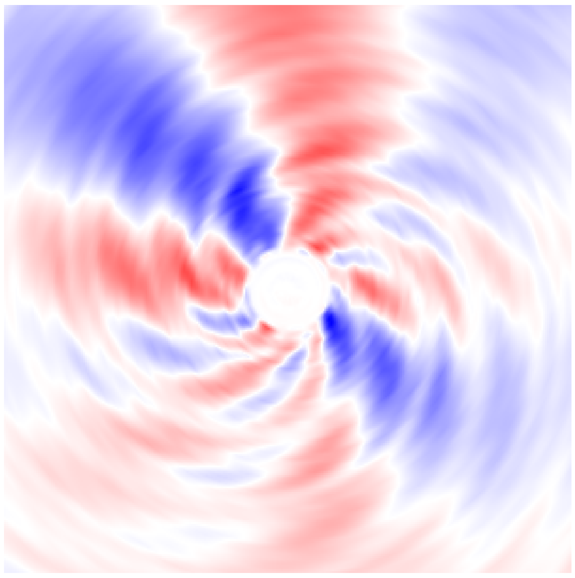


Figure 19: A sound from the same piano, two octaves lower than in figure 17

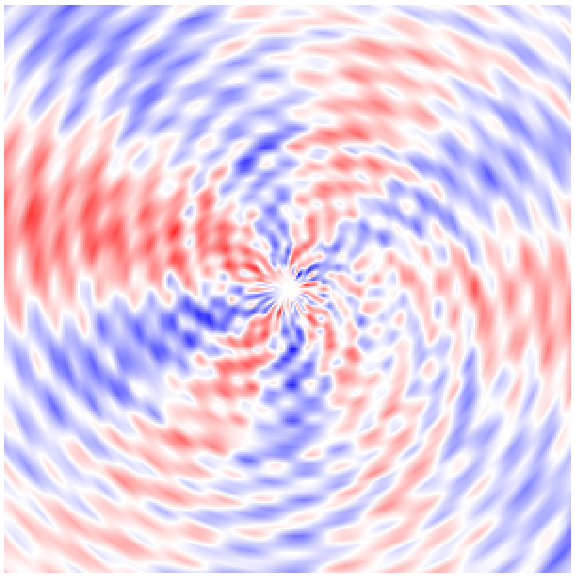


Figure 20: A sound from the same piano, three octaves lower than in figure 17

5 Applications

5.1 The effect of the resonance parameter of an analog filter

The following pictures show a sound produced with a Crumar Bit01 synthesizer. From picture to picture the resonance parameter has been increased by 10 in the units of the synthesizer (no resonance = 0, max. resonance = 63).

The sound is based on a single sawtooth oscillator which is filtered by the synthesizer's 24dB/octave lowpass filter whose cutoff frequency decreases during the evolution of the sound.

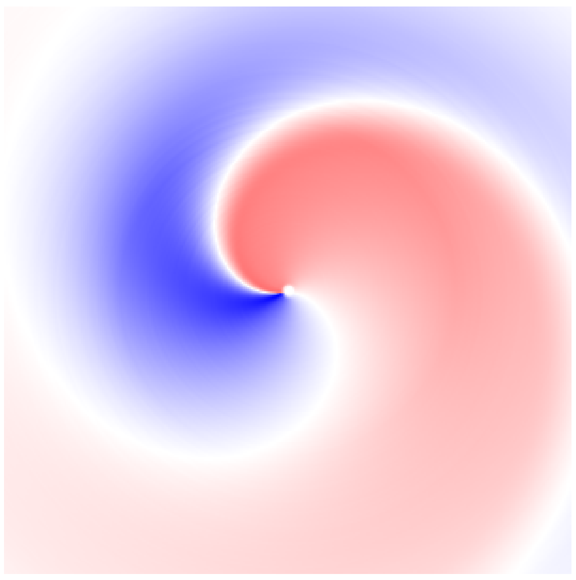


Figure 21: *Res.* = 0; The base frequency could be better adapted, but in order to have the same frequency in all pictures of this sound it is adapted to the base frequency of the sound at higher resonance levels.

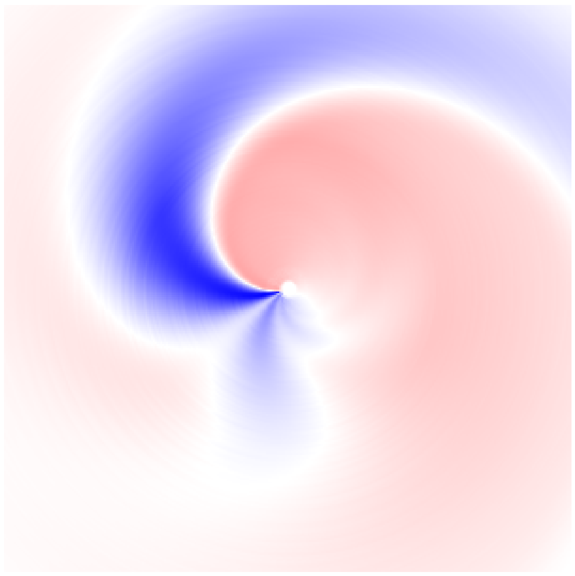


Figure 22: $Res. = 10$; The filter starts to produce a small maximum (the blue streak that points down) which gets more and more delayed while the cutoff frequency decreases.

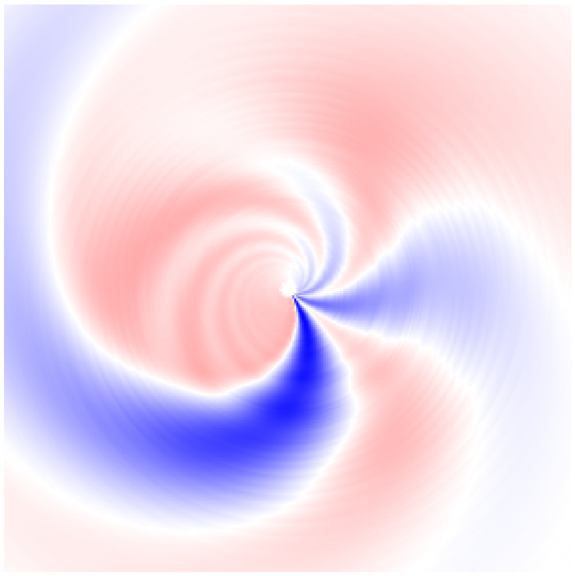


Figure 23: $Res. = 20$; The maximum has become a series of maxima and minima (the red and blue streaks that point to the right and the top).

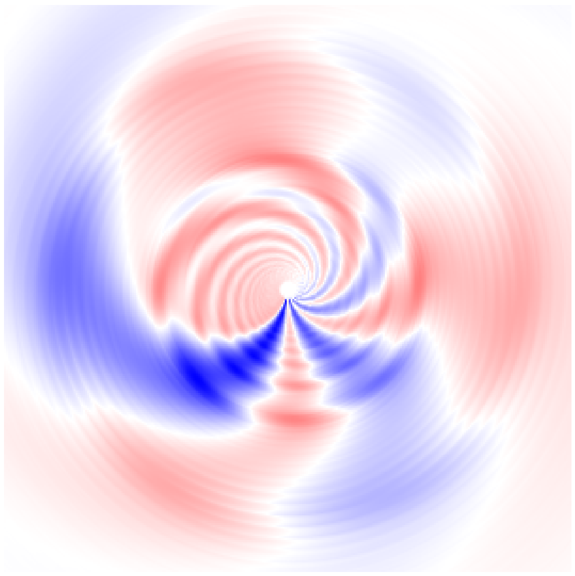


Figure 24: $Res. = 30$; The main maximum and minimum of the sawtooth waveform become hard to distinguish from the "ripples" created by the oscillation of the filter.

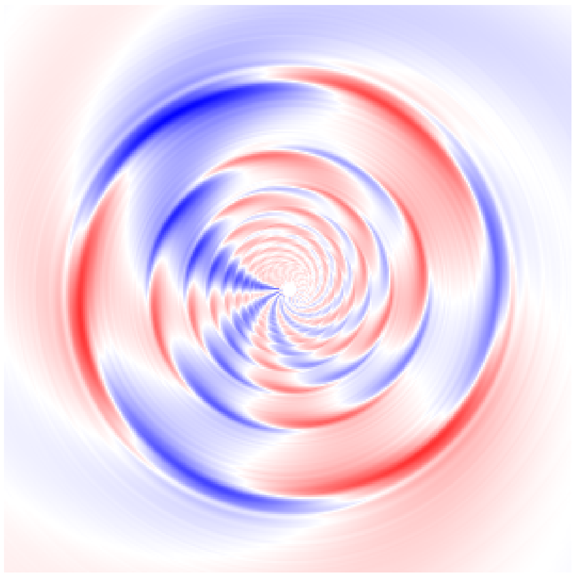


Figure 25: $Res. = 40$; It is nice to see how the filter emphasizes the harmonics of the sawtooth waveform. The outermost "ring" shows the filter oscillating at three times the sawtooth's base frequency ($f_{cutoff} = 3f_{sawtooth}$).

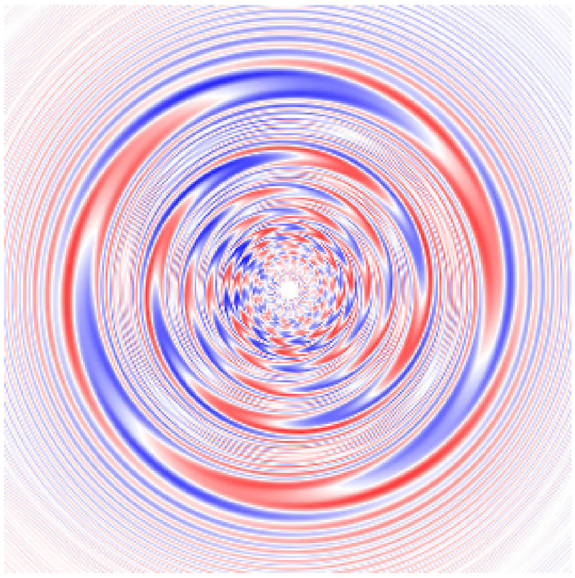


Figure 26: $Res. = 50$; The filter is self-oscillating. Therefore the signal is not pseudo-periodic anymore and it doesn't make much sense to use this method to display it.

5.2 FM synthesis: The effect of the frequency modulation depth

The pictures in this section show a bass sound produced with a Yamaha TX81Z FM synthesizer. From picture to picture the modulation depth of one of the oscillators is increased. The pattern in the last picture is quite typical for FM synthesis. At least I've never seen a pattern like this in a sound from an analog synthesizer.

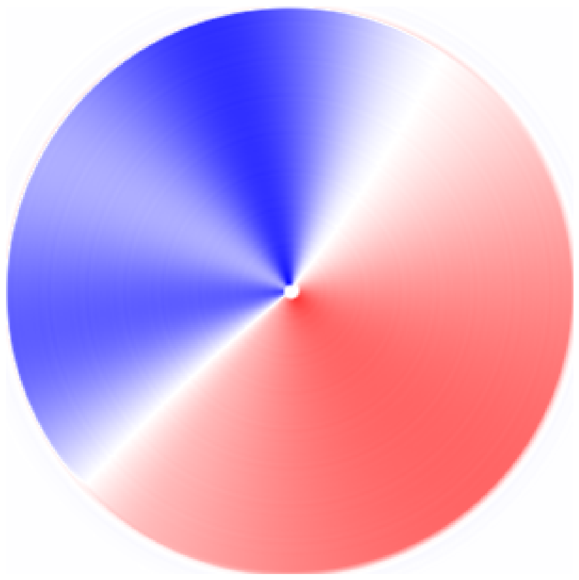


Figure 27: A very boring bass sound

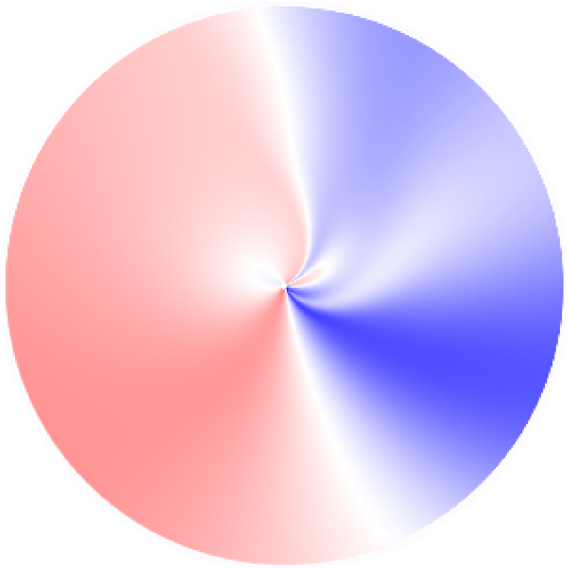


Figure 28: A bit of FM

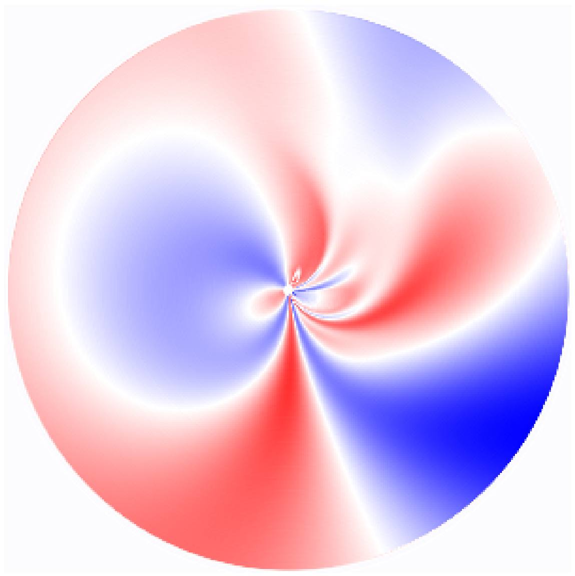


Figure 29: More FM

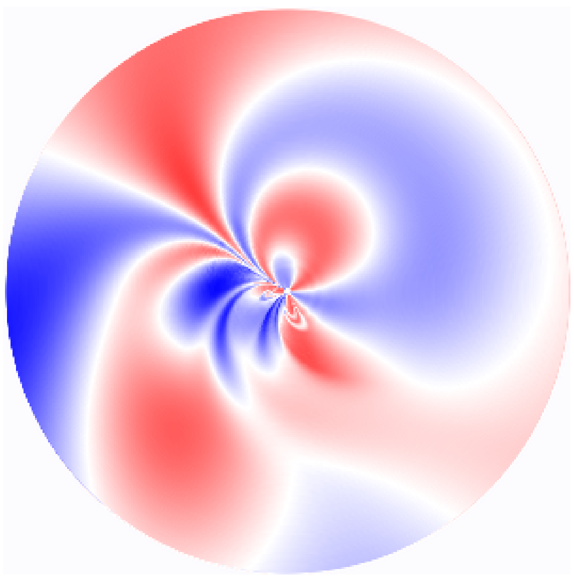


Figure 30: A lot of FM

5.3 Compression algorithm comparison

In this section we look at the attack phase of the sound shown in figure 30 and consider the different artifacts produced by MSADPCM and MP3 compression algorithms at different compression settings.

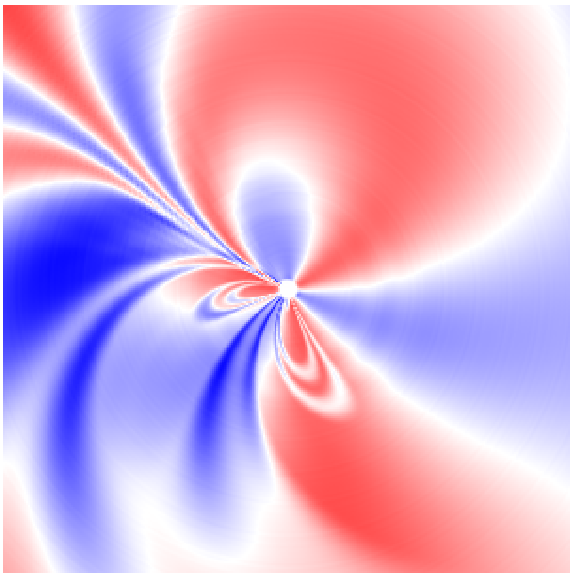


Figure 31: No compression

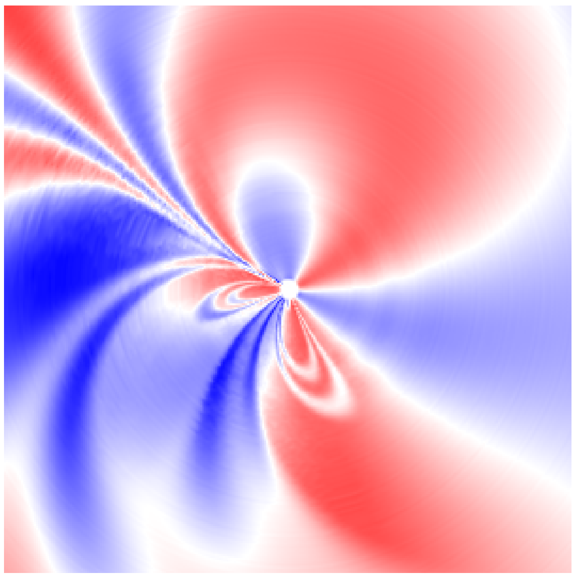


Figure 32: MS ADPCM compression. Some noise (white streaks) visible in the top left corner.

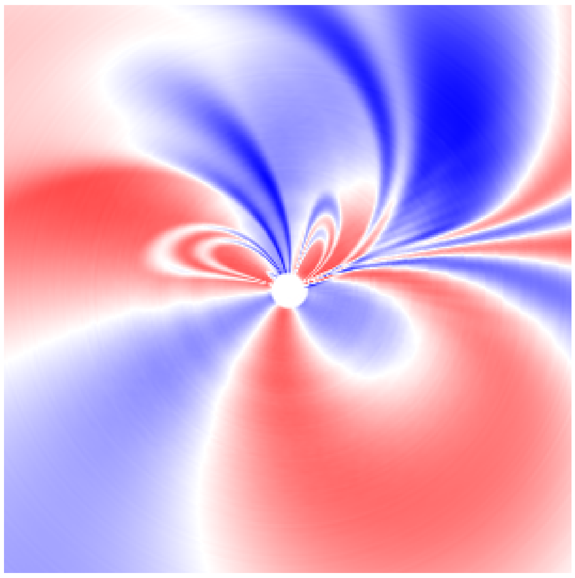


Figure 33: MP3, BladeEnc, 128 kbps. The encoder added some silence (white dot in the middle) in the beginning of the sample which causes the whole picture to be turned.

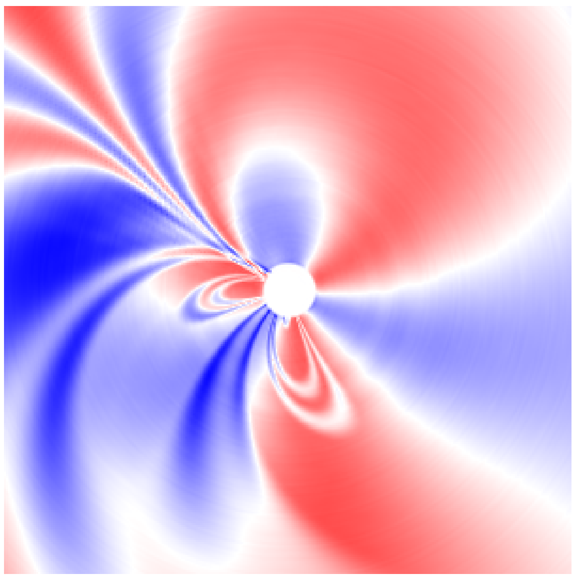


Figure 34: MP3, encoder from Fraunhofer Gesellschaft, 128 kbps. This encoder adds even more silence (white dot even bigger). Almost no artifacts visible.

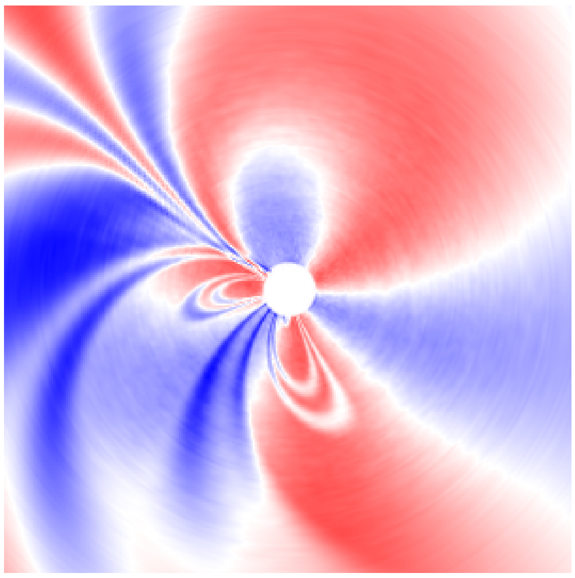


Figure 35: MP3, encoder from Fraunhofer Gesellschaft, 64 kbps.

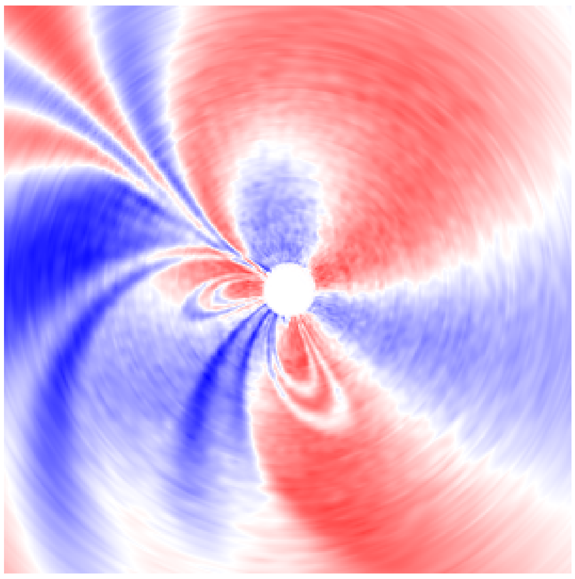


Figure 36: MP3, encoder from Fraunhofer Gesellschaft, 32 kbps.

A Interpolation algorithm

```
#include <math.h>

float *sample;
int32 samplelength=0;
float T;          // samples per period = fs/f
                  // x = alpha * beta * n * cos(beta*n)
                  // y = alpha * beta * n * sin(beta*n)
                  // beta = 2*pi/T
float alphap;     // alpha'=2*pi*alpha = 'step' of spiral


float f(float x, float y)
{
  if (samplelength>0)
  {
    int32 nb,nc;
    float thetap=atan2f(-y,-x)/ZWOPI+0.5; // theta(x,y) <=> atan2f(y,x)
    float m=sqrtf(x*x+y*y)/alphap-thetap;
    float ipm=floorf(m);
    float fpm=m-ipm;
    float nk=(ipm+thetap)*T;
    float nl=nk+T;
    float ipnk=floorf(nk);
    float ipnl=floorf(nl);
    float fpnk=nk-ipnk;
    float fpnl=nl-ipnl;
    nb=(int32)ipnk;
    nc=(int32)ipnl;

    int32 nd=nc+1;
    if (nd<0)
      nd=0;
    else if (nd>=samplelength)
      nd=samplelength-1;
    if (nc<0)
      nc=0;
    else if (nc>=samplelength)
      nc=samplelength-1;

    int32 na=nb+1;
    if (na<0)
      na=0;
    else if (na>=samplelength)
      na=samplelength-1;
    if (nb<0)
      nb=0;
    else if (nb>=samplelength)
      nb=samplelength-1;
    return ((1-fpnk) * sample[nb] + fpnk     * sample[na]) * (1-fpm)
         + (fpnl     * sample[nd] + (1-fpnl) * sample[nc]) * (fpm);
  }
  else return 0;
}
```