Mag. Thomas Mejstrik BA

# Analyse von Musik mit frei wählbaren Zeit-Frequenz Atomen

## Real Time Computation of Redressed Frequency Warped Gabor Expansion

**Diplomarbeit**

zur Erlangung des akademischen Grades
Magister artium

Studium: Masterstudium
Instrumental(Gesangs)pädagogik, (Klavier - Klassik)

Institut 01 Institut für Komposition und Elektroakustik

Universität für Musik und darstellende Kunst Wien

Betreuer: Univ.-Prof. Ph.D. M.S. Gianpaolo Evangelista

Wien 2015

**Abstract**

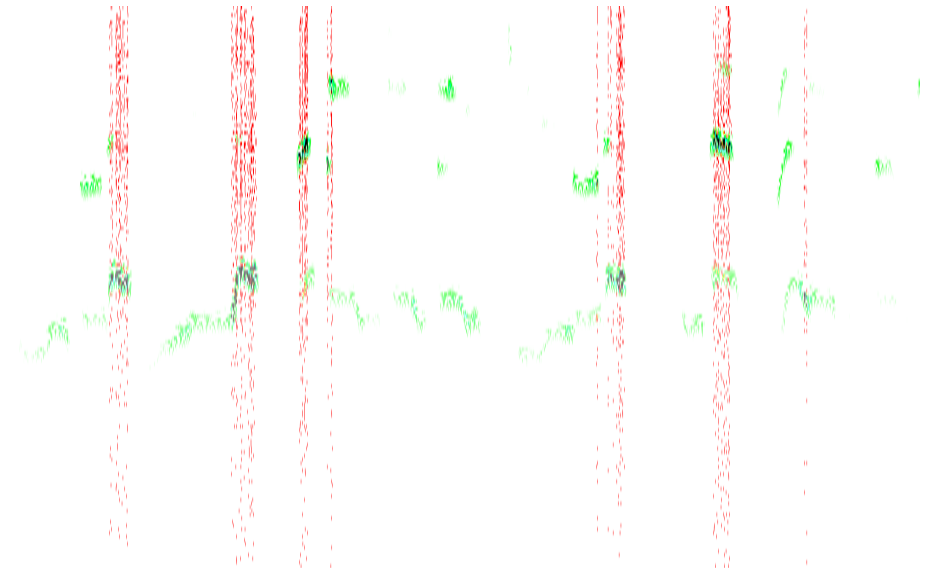
The construction of frames with non-uniform time and frequency resolution can allow us to adapt the atoms to certain properties of the signal to be analysed. Evangelista showed in [7] that this is entirely possible with the sole use of warping operators in the time and frequency domain. In the case of non-uniform frequency tiling of the TF plane, this leads to atoms with unbounded support in the time domain, which renders this approach impossible for real time computation. Evangelista proposes in the same paper two approximations for finite length windows.

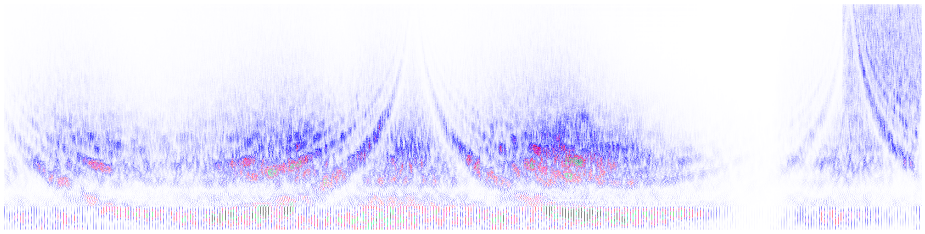
This thesis presents a plug-in for the programming language Pure Data which implements this idea. I show how the implementation works and discuss different possibilities of the implementation always with focus on how to obtain real time computation. After this a short section discusses the computational and memory costs for this algorithm. It ends with measurements of the relative error of an analysed-synthesised signal using different test signals (noise, sinusoids, clicks, music). It turns out that the algorithm in the current stage reaches a relative error of about -57 dB for white noise and a window length of 0,4 s.

**Zusammenfassung**

Die Konstruktion von Frames mit nicht uniformer Zeit und Frequenzauflösung erlaubt es, die Fenster an vorliegende Eigenheiten des zu analysierenden Signals anzupassen. Evangelista zeigt in [7], dass dies möglich ist mit der bloßen Anwendung von Warping Operatoren, die im Zeit und Frequenzbereich arbeiten. Im Falle von nicht gleichförmiger Frequenzaufteilung der TF Ebene führt dies im Allgemeinen zu zeitlich unbeschränkten Fenstern, wodurch Echtzeit Berechnungen unmöglich werden. In der gleichen Arbeit zeigt Evangelista aber auch Approximationen für endliche Fenster.

Diese Arbeit stellt ein Plug-In für die Programmiersprache Pure Data vor die diese Idee implementiert. Ich zeige, wie diese Implementierung funktioniert und bespreche auch andere Möglichkeiten mit dem Augenmerk auf Echtzeitberechnung. Danach folgt ein kurzer Abschnitt über den Berechnungsaufwand und Speicheraufwand dieses Algorithmus. Die Arbeit endet mit Messungen des relativen Fehlers von analysiert-synthetisierten Signalen. Es zeigt sich, dass mit dem Algorithmus in der derzeitigen Form ein relativer Fehler von etwa -57 dB erreicht werden kann für weißes Rauschen und einer Fensterlänge von 0,4 s.





The upper picture is the first spectrogram which was made with my program. It shows the beginning of Susan Vega's *Tom's Diner*. The second picture is the first spectrogram which was made with my program when it finally worked. It shows some swooshes (In this picture, the non uniform data rate of the coefficients is easily visible).

I would like to thank my parents, Olga Podovalova, my professor Gianpaolo Evangelista, my professor Johannes Marian, my professor Manon Liu Winter, Jan, my brother Martin Mejstrik, Katja Link and my teacher Lovorka Schenk who were all involved in this thesis or in my piano studies.

# Contents

# Chapter 1

# Introduction

Music, as physically seen, is just a change of pressure in some medium (probably air). If you want to describe this mathematically, you end up with a function describing the current pressure over time. If you visualize this, you usually do not see much of the music you heard before (e.g. see figure 1.1)

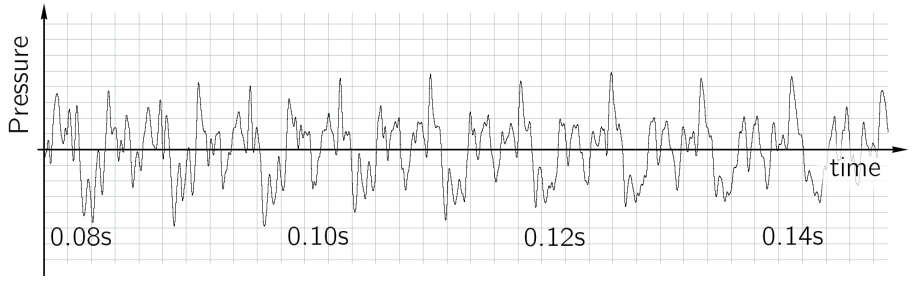



Figure 1.1: Music, visualized as a function of pressure over time.

One method to visualize and analyse sound in a better way, is to plot the spectral components (frequencies contained inside the music) instead of the pressure change. This way our example looks like the plot in figure 1.2. The time is plotted on the x-axis. On the y-axis one can see all the frequencies which are present at a certain time. The more intense red, the higher the magnitude of that frequency. From the image we can see that the sound does not change much over time. It mostly only becomes more and more silent (since everything gets brighter). Furthermore, the present frequencies have all a distance of about 133 Hz from each other, and the deepest present frequency is about 133 Hz. This means, the tone mostly consists only of simple sinusoids[1] whose frequencies are all a multiple of 133 Hz. If one knows, that tones from instruments mostly consists of many simple sinusoids, with frequencies all a multiple of some basic frequency, one can see in that figure, that our example will be some kind of tone played by some kind of instrument. And it really is, namely a C3 played by a Fazioli Grand Piano.

The operation we used to generate that picture is the Short-Time-Fourier transformation (STFT) introduced by Dennis Gábor in the year 1946 [10]. The

[1] Sine waves

STFT belongs to a family called Time-Frequency-Representations. The main idea behind these methods is to decompose sound into small bricks, often called atoms which are somewhat an equivalent to the points in pointillism paintings. Some people compare the atoms also to notes in a (piano-roll) score, but atoms are usually much shorter than any notes in a score. Every atom depicts some sound at a specific time. The differences between these methods is mainly, how these atoms are chosen. Furthermore, one would like to be able to resynthesize the sound from these atoms. Therefore one needs enough atoms to catch all possible sounds. Otherwise some information would be lost.

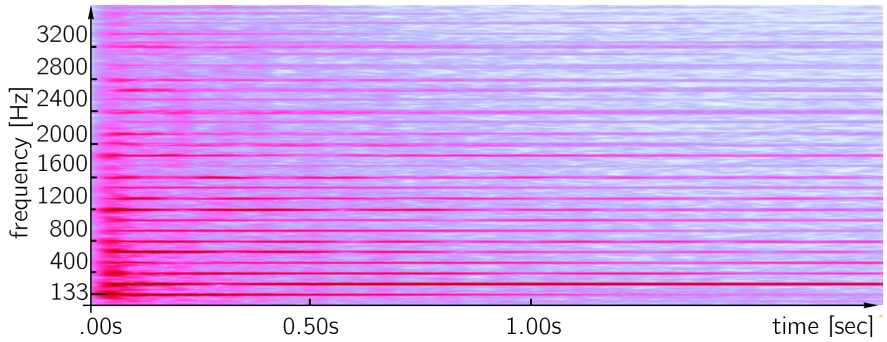



Figure 1.2: Spectrogram of the music in figure1.1.

The representation of music in terms of time and frequency (in the so called *Time-Frequency-Plane*, abbreviated *TF-plane*) has more advantages than just seeming natural to us. For example, one can systematically eliminate or boost certain frequencies (used in denoisers, equalizers, compressors), compress music (used in mp3-files), tune guitars, let the computer typeset musical scores or identify instruments with the computer.

## 1.1 Overview of the content of the thesis

In chapter 2 the concepts of vector space, basis, orthonormal basis, inner product, Fourier Series, continuous Fourier transform, and frames are reviewed. Chapter 3 gives an overview of the Short Time Fourier transform and the Gabor Expansion which we will need later on. Furthermore the Wavelet transform is briefly illustrated. Section 3.4 explains the mathematical background of how to warp Gabor Frames and how to redress them, as it is presented in [7]. In chapter 4, I first discuss which problems arise in the real time computation case. I then present the implementation of my algorithm, divided in the parts atom computing, analysis and synthesis. Additionally, notation, which is used afterwards, is set. Section 4.4 deals with the computational costs of the algorithm. The costs are quoted as number of instructions per sample. The section 4.4 contains measurements of the relative analysis-synthesis error conducted with different test signals and parameters. Chapter 5 gives a very brief summary and a list points of what should be done further. The appendix contains spectrograms from the computational error section and an explanation on how to use the Pure Data-external together with a description of its interface. Links where to download Pure Data and manuals for Pure Data can be found in section 6.3.

# Chapter 2

# Mathematical Preliminaries

Since the basis of most time-frequency representations is the Fourier transform, I first give an overview about this important tool. It was introduced in 1807 by Joseph Fourier[1] but partially used much longer before by Carl Friedrich Gauss, Jean le Rond d'Alembert, Lagrange and even the Babylonians. Fourier had the idea that every periodic function is decomposable into a sum of the most basic periodic functions we know, namely sines and cosines which can are characterized by their fundamental frequency and amplitude.

There are different approaches to get to the Fourier transform. An easy approach, aimed at musicians, can be found in [1]. A more mathematical, but also a "funny to read and free to download as a a pdf"-approach is [14]. In this thesis I am going to present the Fourier transform as a transformation on function spaces. For that purpose I will first present the notion of a vector space and its bases, then proceed to function spaces. Since bases are not well suited for this application we need the concept of frames too, which are a generalisation of bases. This is not the most instructive approach to the Fourier transformation, but I assume that most readers will already be acquainted with the Fourier transform.

## 2.1 Vector spaces and Bases

Because we will only need vector spaces over $\mathbb{R}$ and $\mathbb{C}$, we do not need the most general definition of vector spaces.

**Definition 2.1.** A *vector space* $\mathcal{V}$ over $\mathbb{R}$ ($\mathbb{C}$) is a set together with two operations $+ : (\mathcal{V} \times \mathcal{V}) \to \mathcal{V}$ and $\cdot : (\mathbb{R}\ (\mathbb{C}) \times \mathcal{V}) \to \mathcal{V}$ and that satisfy the following axioms. Let $u, v, w \in \mathcal{V}$ and $a, b \in \mathbb{R}$ ($\mathbb{C}$).

- $v + w = w + v$
- $(u + v) + w = u + (v + w)$
- There exists $\mathcal{O} \in \mathcal{V}$ such that $v + \mathcal{O} = v$ for all $v \in \mathcal{V}$
- For all $v \in \mathcal{V}$ there exists $-v \in \mathcal{V}$ such that $v + (-v) = \mathcal{O}$
- $a(b \cdot v) = (ab) \cdot v$
- $1 \cdot v = v$ for all $v \in \mathcal{V}$ and $1 \in \mathbb{R}$
- $a(u + v) = au + av$

[1] Jean Baptiste Joseph Fourier; * 21. March 1768 at Auxerre, † 16. May 1830 in Paris.

- $(a+b)\cdot v = a\cdot v + b\cdot v$

The writing of the "$\cdot$" is often omitted. Elements of $\mathcal{V}$ will be called *vectors*, elements of $\mathbb{R}$ ($\mathbb{C}$) will be sometimes called *scalar*.

This definition looks complicated, but altogether it states that one can add vectors, scale vectors and that there exists something like the number 0. Some examples will make things clear.

**Example 2.2.** The most basic example of a vector space is the $\mathbb{R}^2$, the plane. Let $\mathcal{V} = \{(x,y)|x,y\in\mathbb{R}\}$. We define $(x_1,y_1)+(x_2+y_2) = (x_1+x_2, y_1+y_2)$ and for a number $a\in\mathbb{R}$ let $a(x,y) = (ax,ay)$. This defines a vector space, which is easily checked, let $u=(x_1,y_1)$, $v=(x_2,y_2)$, $w=(x_3,y_3)$ and $a,b\in\mathbb{R}$.

- $v+w = (x_2+x_3, y_2+y_3) = (x_3+x_2, y_3+y_2) = w+v$
- $(u+v)+w = (x_1+x_2, y_1+y_2)+(x_3,y_3) = (x_1+x_2+x_3, y_1+y_2+y_3) = u+(v+w)$
- $v+(0,0) = (x_2+0, y_2+0) = (x_2,y_2) = v$ and $(0,0)\in\mathcal{V}$, hence $(0,0)=\mathcal{O}$
- Let $-v = (-x_2,-y_2)\in\mathcal{V}$, then $v+(-v) = (0,0) = \mathcal{O}$
- $1v = 1(x_2,y_2) = (x_2,y_2) = v$
- $a(u+v) = a(x_1+x_2, y_1+y_2) = (ax_1+ax_2, ay_1+ay_2) = au+av$
- $(a+b)v = ((a+b)x_2, (a+b)y_2) = (ax_2+bx_2, ay_2+by_2) = av+bv$

Since all conditions are fulfilled, $\mathbb{R}^2$ is a vector space. Graphically, $\mathbb{R}^2$ can be imagined as a space consisting of arrows. Adding vectors corresponds to pasting arrows *stem on point* together. Multiplying vectors with scalars corresponds to scaling the arrows, see figure 2.1. The same calculations work for any $\mathbb{R}^n$ with $n$ finite. Hence all $\mathbb{R}^n$ with $n<\infty$ are vector spaces (nothing is said in the case $n=\infty$ here).

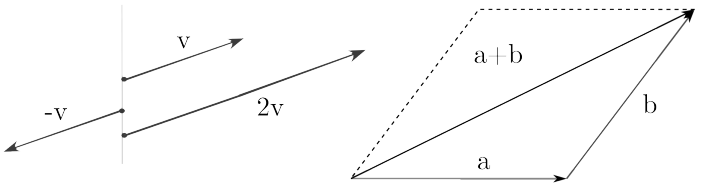



Figure 2.1: Left hand side: The vector $v$ together with scaled versions $2v$ and $-v$. Right hand Side: The vectors $a$ and $b$ and their sum $a+b$.

The next example will not be as well-known, but after the definitions it will be as obvious that it constitutes a vector space.

**Example 2.3.** Let $\mathscr{F}(\mathbb{R}\to\mathbb{R})$ denote all functions from $\mathbb{R}$ to $\mathbb{R}$. If $f,g$ are two functions in $\mathscr{F}(\mathbb{R}\to\mathbb{R})$ and $a\in\mathbb{R}$ ($\mathbb{C}$) then we define $(f+g)(x) := f(x)+g(x)$ and $(a\cdot f)(x) := af(x)$. It is very easy to check all the vector space axioms. I will only proof the third and fourth condition here

- Let $\mathcal{O}(x) := 0$ for all $x\in\mathbb{R}$, then $(f+\mathcal{O})(x) = f(x)+\mathcal{O}(x) = f(x)+0 = f(x)$. Hence $\mathcal{O}$ is the zero in the function space $\mathscr{F}(\mathbb{R}\to\mathbb{R})$.
- Let $g(x) := -f(x)$, then $(f+g)(x) = f(x)+g(x) = f(x)+(-f(x)) = f(x)-f(x) = 0 = \mathcal{O}(x)$. Hence $g$ is the negative vector to $f$.

$\mathscr{F}(\mathbb{R}\to\mathbb{R})$ (probably) can not be imagined like the $\mathbb{R}^2$ with vectors. Instead this vector space is easily imaginable if on thinks how the two operations work on the functions. "+" adds two functions pointwise, "$\cdot$" scales a function.

**Definition 2.4.** A vector space which consists of functions is called *function-space.*

**Example 2.5.** Further examples of vector spaces are:

- The set $\ell$ of series $(a_0, a_1, a_2, a_3, \cdots)$ with $a_i \in \mathbb{R}$ $(\mathbb{C})$. This vector space can be seen as an infinite dimensional generalisation of the $\mathbb{R}^2$ $(\mathbb{C}^2)$ vector space above.
- The complex numbers $\mathbb{C}$. These can be seen as a vector space over $\mathbb{R}$.
- The polynomials $\mathscr{P}^n = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$, either with finite degree $n < \infty$, but also with infinite degree $n = \infty$. This vector space can be seen as an finite dimensional $\mathbb{R}^n$ if $n < \infty$ or as the space of series $\ell$ if $n = \infty$.

In some vector spaces we can define a function which measures vectors, e.g. determines their length. These spaces are called Banach spaces.

**Definition 2.6.** Let $\mathcal{V}$ be a vector space together with a function called *norm* $||\cdot|| : \mathcal{V} \to \mathbb{R}$ which assigns a real number to every vector. The norm must have the following properties, let $v, w \in \mathcal{V}$:

- If $||v|| = 0$ then $v = \mathcal{O}$
- Absolute scalability: $||av|| = |a|\,||v||$ for every $a \in \mathbb{R}$ $(\mathbb{C})$
- Triangle inequality: $||v + w|| \leq ||v|| + ||w||$

If the space is complete (i.e. every Cauchy sequence converges) then this space is called a *Banach space.*

**Example 2.7.** A norm for $\mathbb{R}^2$ is $(x, y) \mapsto \sqrt{x^2 + y^2}$. Other norms would be $(x, y) \mapsto |x| + |y|$ and $(x, y) \mapsto \max\{|x|\,, |y|\}$.

Some vector spaces can be drawn using a coordinate system, like the $\mathbb{R}^2$ or $\mathbb{R}^3$. A coordinate system consists of axes, which are the directions in which the vector space extends. This directions are actually also vectors of the vector space. If we generalize this concept, we arrive at the concept of (Schauder-)bases.

**Definition 2.8.** Let $\mathcal{V}$ be a Banach space. Let $\mathcal{B} = \{b_1, b_2, \cdots\} \subset \mathcal{V}$ be a (particularly countable) subset of $\mathcal{V}$. If every vector $v \in \mathcal{V}$ can be written as a convergent sum (convergent in the norm of the space)

$$v = \alpha_1 b_1 + \alpha_2 b_2 + \cdots \quad \text{with} \quad \alpha_1, \alpha_2, \cdots \in \mathbb{R}\ (\mathbb{C}) \tag{2.1}$$

and this representation is unique (i.e. if one has two representations $v = \alpha_1 b_1 + \alpha_2 b_2 + \cdots = \beta_1 b_1 + \beta_2 b_2 + \cdots$ with $\alpha_i, \beta_i \in \mathbb{R}$ $(\mathbb{C})$ then $\alpha_i = \beta_i$ for all $i$) than $\mathcal{B}$ is a *(Schauder-)basis* for $\mathcal{V}$. The numbers $\alpha_i$ written as the sequence $(\alpha_1, \alpha_2, \alpha_3, \cdots)$ are called the *coordinates of $v$ in the basis $\mathcal{B}$.*

This condition has the following consequences

- If one vector is removed from $\mathcal{B}$ then there are vectors $v \in \mathcal{V}$ which cannot be written as the sum (2.1). That means that really all vectors in $\mathcal{B}$ are needed to generate the vector space $\mathcal{V}$.
- If one vector is added to $\mathcal{B}$, then the uniqueness in the definition does not hold any more.
- If $\alpha_1 b_1 + \alpha_2 b_2 + \cdots = 0$ then all $\alpha_i = 0$ (which follows directly from the uniqueness).

The number of elements in $\mathcal{B}$ does not need to be finite. If it is finite, then all bases have the same quantity of elements. This number of vectors is called the dimension of the vector space $\mathcal{V}$. If it is infinite, then all bases have infinite elements and we say the dimension is $\infty$.

To understand vector spaces and bases better we will look at the above examples again.

**Example 2.9.** A basis for $\mathbb{R}^2$ can be $\mathcal{B} = \{(1,0),(0,1)\}$. Proof: Let $v = (x,y) \in \mathcal{V}$. We must be able to write $v$ as a sum of vectors in $\mathcal{B}$. $v = x(1,0) + y(0,1) = (x,0) + (0,y) = (x,y)$. Check. Since this is the only solution, the representation is unique too. This basis is called *standard basis* for $\mathbb{R}^2$.

If we add one vector $(1,1)$ to the basis $\mathcal{B}$, then $\mathcal{B}$ cannot be a basis any more according to the previous remark. Indeed, let $v = (2,3)$. Then we can write $v = 2 \cdot (1,1) + 1 \cdot (0,1)$ but also $v = 2 \cdot (1,0) + 3 \cdot (0,1)$.

If we remove one vector, for example the second, there are vectors which cannot be represented any more. For example, let $v = (0,4)$. We would need $(0,4) = \alpha(1,0)$, hence we have the two equations

$$0 = \alpha \cdot 1$$
$$4 = \alpha \cdot 0$$

This can never be true, hence the set $\{(1,0)\}$ is no basis for $\mathbb{R}^2$ any more.

In figure 2.2 the point P is drawn in the $\mathbb{R}^2$ plane and two bases are used to express the coordinates of the point. The first is the standard basis $\mathcal{B} = \{b_1, b_2\}$, $b_1 = (0,1)$ and $b_2 = (1,0)$. The second basis is denoted with $\mathcal{C} = \{c_1, c_2\}$ where $c_1 = (2,4), c_2 = (0,-1)$. The basis vectors are drawn in the left coordinate system. In the right coordinate system we describe the point $P = (2,2)$ in two different bases. We first see (the blue arrows): $2b_1 + 2b_2 = 2 \cdot (1,0) + 2 \cdot (0,1) = (2,0) + (0,2) = (2,2)$. Hence $P$ has coordinates $P_{\mathcal{B}} = (2,2)$ in basis $\mathcal{B}$. In Basis $\mathcal{C}$ (the green arrows) the point is expressed as: $1c_1 - 2c_2 = 1 \cdot (2,4) - 2 \cdot (0,-1) = (2,4) - (0,2) = (2,2)$. Hence the Peach has coordinates $P_{\mathcal{C}} = (1,-2)$ in basis $\mathcal{C}$. That shows, different bases yield to different representations of the same thing.

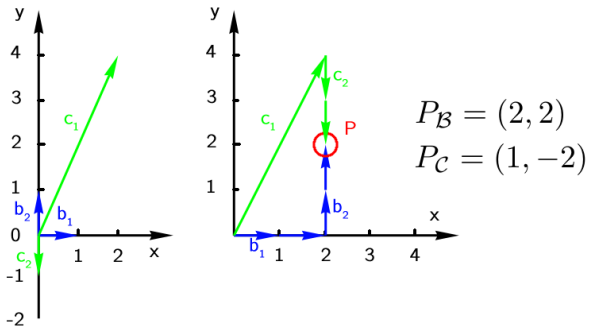



Figure 2.2: The point P in two different bases $\mathcal{B} = \{b_1, b_2\}$ (blue) and $\mathcal{C} = \{c_1, c_2\}$ (green).

**Example and Definition 2.10.** Function spaces $\mathscr{F}$ do not have a Schauder-basis in general. If we restrict ourselves to functions with certain properties, there are spaces with a Schauder-basis.

Let $L^2[0, 2\pi)$ be the *space of square integrable functions*, i.e.

$$\int_0^{2\pi} |f(x)|^2 \, dx < \infty$$

*from the interval* $[0, 2\pi)$ *to* $\mathbb{R}$ *(*$\mathbb{C}$*).* This space has a norm which can be defined as

$$||f|| = \sqrt{\int_0^{2\pi} |f(v)|^2 \, dx}.$$

The set of functions $A_k(x)$ and $B_k(x)$

$$\begin{aligned} A_k(x) &= \frac{1}{\pi}\cos(kx) \text{ with } k \in \mathbb{N} \cup \{0\} \\ B_k(x) &= \frac{1}{\pi}\sin(kx) \text{ with } k \in \mathbb{N} \setminus \{0\} \end{aligned} \tag{2.2}$$

forms a basis (with this norm) (a proof can be found in any book about harmonic analysis). These are infinitely many functions, hence the vector space is infinite dimensional. We will see these functions again, because they are the core of the Fourier transform.

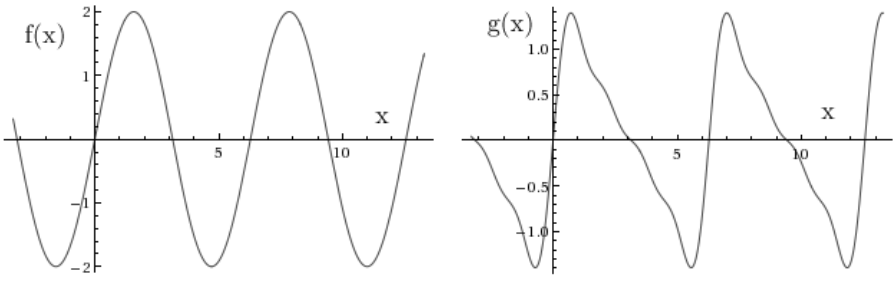



Figure 2.3: Left hand side: The function $f$ with coordinate sequence $(\beta_k)_{k>0} = (2, 0, 0, \cdots)$ and all coefficients $\alpha_i$ for the $A_i$ functions zero. I.e. $f(x) = 2\sin(x)$. Right hand side: The function $g$ with coordinate sequence $(\beta_k)_{k>0} = (1, 0.5, 0.25, 0.125, 0, 0, \cdots)$ and all coefficients $\alpha_i$ for the $A_i$ functions zero. I.e. $g(x) = sin(x) + 0.5 * sin(2x) + 0.25 * sin(3x) + 0.125 * sin(4x)$

**Example 2.11.**

- A basis for the vector space $\mathbb{C}$ over $\mathbb{R}$ is $\mathcal{B} = \{1, i\}$.
- A basis for the sequence space $\ell$ is the set $\mathcal{B} = \{e_k | k > 0\}$. The $e_k$ are the sequences $(0, 0, \cdots, 1, 0, 0 \cdots)$ where the *1* stands on the $k^{th}$ position.

In some vector spaces ($\mathbb{R}^2$, $\mathbb{R}^3$) we can measure lengths and angels. This is very useful since it allows us to define what orthogonality means. I will skip the motivation why length and angels can be measured that way.

**Definition 2.12.** Let $\mathcal{V}$ be a vector space. An operation $\langle \cdot, \cdot \rangle : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ ($\mathbb{C}$) taking two vectors of $\mathcal{V}$ and associating them to a real number (complex number) is an *inner product* (also *scalar product* and *dot product*) if the following conditions hold, let $v_1, v_2 \in \mathcal{V}$:

- Conjugate symmetric: $\langle v_1, v_2 \rangle = \overline{\langle v_2, v_1 \rangle}$. The overline denotes complex conjugation. If the inner product is real, then the conjugation has no effects.
- Linear in the first argument: $\langle a v_1 + v_2, v_3 \rangle = \langle a v_1, v_3 \rangle + \langle v_2, v_3 \rangle$. Together with the first condition it follows that it is conjugate linear in the second argument.[2]
- $\langle v, v \rangle \geq 0$ for all $v \in \mathcal{V}$.
- If $\langle v, v \rangle = 0$ then $v$ must be $\mathcal{O}$.

With the inner product one can define a norm via, let $v \in \mathcal{V}$:

$$||v|| := \sqrt{\langle v, v \rangle}$$

This fulfils all conditions needed for a norm. If one thinks of the inner product as a product with real numbers then $||v|| = \sqrt{\langle v, v \rangle} = \sqrt{v \cdot v} = \sqrt{v^2} = |v|$. Thus in the case of simple numbers, the norm and the absolute value are the same.

The *angel* $\sphericalangle$ between two vectors $v_1, v_2$ is defined as

$$\sphericalangle(v_1, v_2) = \arccos\left(\frac{|\langle v_1, v_2 \rangle|}{||v_1||\,||v_2||}\right)$$

If the vector space is real, then we do not need the absolute value in the fraction. If the inner product in the formula $\langle v_1, v_2 \rangle$ is zero, than the *arccos* is 90°. Hence we define that two vectors are *normal* onto each other if their inner product is zero.

**Definition 2.13.** A vector space with an inner product which is complete (i.e. every Cauchy sequence converges) is called *Hilbert space*.

The inner product induces the concept of geometry into the vector space. The same vector space can have different geometries by using different inner products. The inner product is a very special property and there are a lot of vector spaces where one cannot find one. But fortunately the spaces we need for the Fourier transformation have one. We examine again our examples.

**Example 2.14.** The space $\mathbb{R}^2$: Let $v_i = (x_i, y_i)$. The map $\langle v_1, v_2 \rangle := x_1 x_2 + y_1 y_2$ defines an inner product. Proof:

- Symmetry: $\langle v_1, v_2 \rangle = x_1 x_2 + y_1 y_2 = x_2 x_1 + y_2 y_1 = \langle v_2, v_1 \rangle$
- Linearity: $\langle a v_1 + v_2, v_3 \rangle = (a x_1 + x_2) x_3 + (a y_1 + y_2) = \cdots = a \langle v_1, v_3 \rangle + \langle v_2, v_3 \rangle$
- $\langle v, v \rangle = xx + yy = x^2 + y^2 > 0$
- $0 = \langle v, v \rangle = x^2 + y^2 = 0 \Rightarrow x = 0$ and $y = 0$.

The length of the vector $v = (1, 2)$ is $||v|| = \sqrt{1^2 + 2^2} = \sqrt{5}$. The angle between $v_1 = (1, 2)$ and $v_2 = (4, -3)$ is

$$\sphericalangle(v_1, v_2) = \arccos\left(\frac{|-6|}{\sqrt{5}\sqrt{25}}\right) \simeq \arccos(0,537\cdots) \simeq 57,5^\circ$$

[2]Sometimes linearity is defined for the second argument.

For this inner product the formulas for lengths and angels are the same as the formulas learned in school. Hence the geometry of $\mathbb{R}^2$ with this inner product is the usual geometry.

The map $\langle v_1, v_2\rangle := x_1x_2 + 2y_1y_2$ also defines an inner product. Using this inner product the vectors $(0,1)$ and $(1,0.5)$ are normal onto each other, but are not normal in the space with the former inner product.

$\mathbb{R}^2$ with any of these inner products is complete, therefore they are a Hilbert space.

**Remark 2.15.** Since the inner product $\langle x, y\rangle$ in $\mathbb{R}^n$ equals the projection of $y$ onto $x$ times the length of $x$ the inner product can be seen as a projection.

**Example 2.16.** The function spaces: Not all function spaces have an inner product. The $L^2[0, 2\pi)$ has one and is complete (without proof), hence it is a Hilbert spaces. Let $f, g \in \mathrm{L}^2[0, 2\pi)$, then

$$\langle f, g\rangle = \int_0^{2\pi} f(x)\overline{g(x)}\,dx$$

is an inner product. If the functions are real valued, the complex conjugation can be omitted.

Since continuous function over a bounded interval are square integrable, we can define an inner product for the functions $A_i$ and $B_i$ defined in 2.2. We are going to calculate the length of these vectors and the angels between them. We start with a recapitulation of some trigonometric identities. Let $k, l \in \mathbb{N} \setminus \{0\}$ (The calculations are trivial respective not needed for zero).

$$\cos((k+l)x) = \cos(kx)\cos(lx) - \sin(kx)\sin(lx) \tag{2.3}$$
$$\sin((l+k)x) = \sin(lx)\cos(kx) + \cos(lx)\sin(kx) \tag{2.4}$$

This adds up to

$$\cos((k+l)x) + \cos((k-l)x) = 2\cos(kx)\cos(lx) \tag{2.5}$$
$$\sin((l+k)x) + \sin((l-k)x) = 2\sin(lx)\cos(kx) \tag{2.6}$$

From this it follows that

$$\begin{aligned}
\int_0^{2\pi} \cos(kx)\cos(lx)\,dx &= \frac{1}{2}\int_0^{2\pi} \Big(\cos((k+l)x) + \cos((k-l)x)\Big)\,dx \\
&= \frac{1}{2(k+l)}\sin((k+l)x)\Big|_0^{2\pi} + \frac{1}{2(k-l)}\sin((k-l)x)\Big|_0^{2\pi} \\
&= \frac{1}{2(k+l)}\Big(\sin\big((k+l)2\pi\big) - \sin\big(0\big)\Big) + \\
&\quad \frac{1}{2(k-l)}\Big(\sin\big((k-l)2\pi\big) - \sin\big(0\big)\Big) \\
&= 0 \quad \text{for all} \quad k \neq l
\end{aligned}$$

Since $\sin(x) = \cos(x + \pi/2)$ the same applies to $\langle B_k, B_l\rangle$. A similar calculation using (2.6) shows

$$\int_0^{2\pi} \cos(kx)\sin(lx)\,dx = 0 \quad \text{for all} \quad k \neq l$$

If $l = k$ and using (2.6) we get $\sin(2kx) = 2\sin(kx)\cos(kx)$ and therefore

$$\int_0^{2\pi} \cos(kx)\sin(kx)\,dx = \frac{1}{2}\int_0^{2\pi} \sin(2kx)\,dx = 0$$

At last we compute (using (2.5)).

$$\int_0^{2\pi} \cos(kx)^2\,dx = \frac{1}{2}\int_0^{2\pi} \cos(2kx) + 1\,dx = \frac{0}{2} + \frac{2\pi}{2} = \pi$$

the same holds for $\sin(kx)$. Summing up we get for $k, l > 0$ (Remember the $A_k$ and $B_k$ functions have a prefactor of $1/\pi$).

$$\begin{aligned}
\langle A_k, A_l\rangle &= \int_0^{2\pi} \cos(kx)\cos(lx)\,dx = \begin{cases} 0 & \text{if } k \neq l \\ 1 & \text{if } k = l \end{cases} \\
\langle A_k, B_l\rangle &= \int_0^{2\pi} \cos(kx)\sin(lx)\,dx = 0 \\
\langle B_k, B_l\rangle &= \int_0^{2\pi} \sin(kx)\sin(lx)\,dx = \begin{cases} 0 & \text{if } k \neq l \\ 1 & \text{if } k = l \end{cases} \\
\langle A_0, A_0\rangle &= 2
\end{aligned}$$

That means, all basis vector are mutually normal onto each other. This is a quite astonishing property. In figure 2.4 we can see some of the products $A_k(x) \cdot B_l(x)$, which occur in the integrals, plotted. It is easily spotted that the integral of these functions is likely to be zero for two different basis vectors and nonzero for the same basis vectors.

**Example 2.17.** Let $\ell^2$ be the space of series $c \in \ell^2$ such that $\sum_{i=0}^{\infty} |c_i|^2 < \infty$ then

$$\langle c_1, c_2\rangle = \sum_{i=0}^{\infty} c_1 \cdot \overline{c_2}$$

is an inner product. It is easily seen that the basis vectors $e_k$ from example 2.9 have the property that

$$\langle e_k, e_k\rangle = \begin{cases} 0 & \text{if } k \neq l \\ 1 & \text{if } k = l \end{cases}$$

The property that all basis vectors are mutually normal onto each other is very useful, hence bases with such a property have a special name.

**Definition 2.18.** Let $\mathcal{B}$ be a basis for the vector space $\mathcal{V}$. If the basis vectors $b_i$ in $\mathcal{B}$ have the property that[3]

- all $b_i$ are normal on each other, i.e. $\langle b_i, b_j\rangle = 0 \quad \forall i \neq j$,
- all $b_i$ have a length of 1, i.e. $||b_i|| = 1 \quad \forall i$,

then the basis is called an *orthonormal* basis (*ONB*), if only the first property holds, the basis is called an *orthogonal* basis. Orthogonal bases can be easily made orthonormal via scaling the basis vectors about their length.

[3]The charater $\forall$ means "for all".

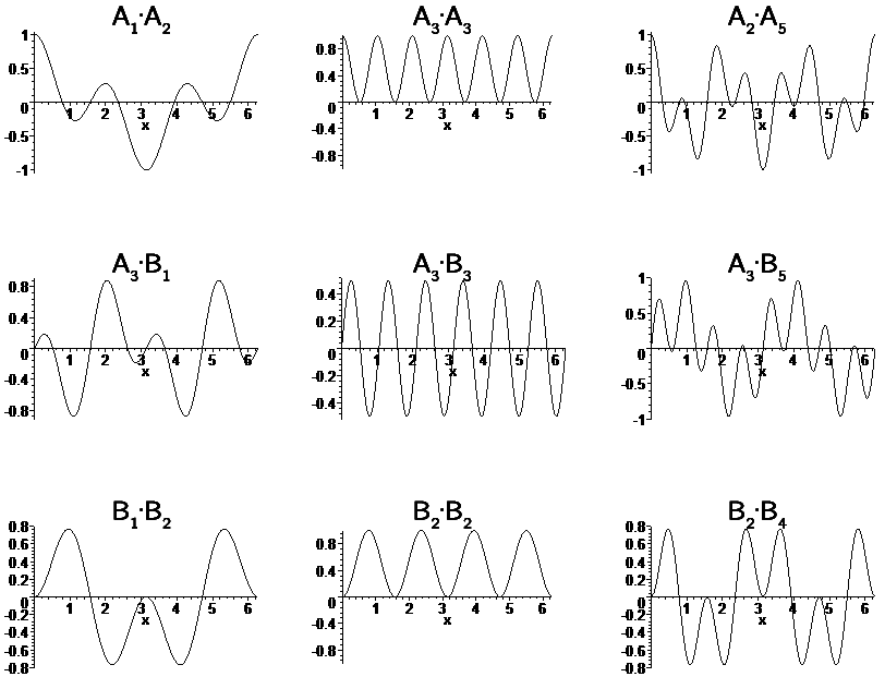



Figure 2.4: Plotted are various products of functions from the basis defined in(2.2). Only the functions $A_3 \cdot A_3$ and $B_2 \cdot B_2$ have an integral unequal zero, and hence $\langle A_3, A_3\rangle, \langle B_2, B_2\rangle \neq 0$.

With this nomenclature the basis for the above given function space $L^2[0, 2\pi)$ with basis functions $A_k$ and $B_k$ is only an orthogonal basis, because the vector $A_0$ has length 2. The given basis for $\ell^2$ is an orthonormal basis. The standard basis for $\mathbb{R}^2$ is also an orthonormal basis.

**Remark 2.19.** If $\mathcal{B} = \{b_1, b_2, b_3, \cdots\}$ is an ONB the coordinates of a vector $v$ in this basis can be computed easily. They are $\langle v, b_i\rangle$ and the vector can be written as

$$v = \sum_{i=0}^{\infty} \langle v, b_k\rangle b_k \tag{2.7}$$

since, let $v = \sum_{i=0}^{\infty} \beta_i b_i$ (this representation exists since $\mathcal{B}$ is a basis). If we compute $\langle v, b_k\rangle = \langle \sum_{i=0}^{\infty} \beta_i b_i, b_k\rangle = \sum_{i=0}^{\infty} \beta_i \langle b_i, b_k\rangle = \beta_k$. Hence $\langle v, b_k\rangle$ is exactly $\beta_k$ and therefore $v = \sum_{i=0}^{\infty} \langle v, b_k\rangle b_k$.

**Example 2.20.** This example is another orthonormal basis for $\mathbb{R}^2$. We consider the basis $\mathcal{B} = \{b_1, b_2\}$, $b_1 = (1/\sqrt{2}, 1/\sqrt{2})$, $b_2 = (1/\sqrt{2}, -1/\sqrt{2})$. We first check that the basis vectors are orthonormal.

$$\begin{aligned}
\langle b_1, b_1\rangle &= \langle (1/\sqrt{2}, 1/\sqrt{2}), (1/\sqrt{2}, 1/\sqrt{2})\rangle = \tfrac{1}{2} + \tfrac{1}{2} = 1 \\
\langle b_1, b_2\rangle &= \langle (1/\sqrt{2}, 1/\sqrt{2}), (-1/\sqrt{2}, 1/\sqrt{2})\rangle = \tfrac{-1}{2} + \tfrac{1}{2} = 0 \\
\langle b_2, b_2\rangle &= \langle (-1/\sqrt{2}, 1/\sqrt{2}), (-1/\sqrt{2}, 1/\sqrt{2})\rangle = \tfrac{1}{2} + \tfrac{1}{2} = 1
\end{aligned}$$

We test equation (2.7) with the vector $v = (2, 3)$. If we compute the inner product of $v$ with the basis vectors we will get the coordinates of $v$ in basis $\mathcal{B}$. I will denote the coordinates of $v$ with $v_{\mathcal{B},1}$ and $v_{\mathcal{B},2}$.

$$\begin{aligned} v_{\mathcal{B},1} &= \langle v, b_1 \rangle = \langle (2,3), (1/\sqrt{2}, 1/\sqrt{2}) \rangle = \tfrac{2}{\sqrt{2}} + \tfrac{3}{\sqrt{2}} = \tfrac{5}{\sqrt{2}} \\ v_{\mathcal{B},2} &= \langle v, b_2 \rangle = \langle (2,3), (1/\sqrt{2}, -1/\sqrt{2}) \rangle = \tfrac{2}{\sqrt{2}} + \tfrac{-3}{\sqrt{2}} = \tfrac{-1}{\sqrt{2}} \end{aligned}$$

To test if this is right, we enter the coordinates and compute the sum of the coordinates with the basis vectors. Then we will get the original vector $v$.

$$\begin{aligned} v_{\mathcal{B},1} b_1 + v_{\mathcal{B},2} b_2 &= \tfrac{5}{\sqrt{2}}(1/\sqrt{2}, 1/\sqrt{2}) + \tfrac{-1}{\sqrt{2}}(1/\sqrt{2}, -1/\sqrt{2}) \\ &= (5/2, 5/2) + (-1/2, 1/2) = (4/2, 6/2) \\ &= (2, 3) = v \end{aligned}$$

**Example 2.21.** Since we already know, that the functions $A_k$ and $B_k$ are an orthogonal basis for $L^2[0, 2\pi)$, we can write all functions $f \in L^2[0, 2\pi)$ as (due to equation (2.7)):

$$\begin{aligned} f(x) =& \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k A_k(x) + b_k B_k(x) \quad \text{with} \\ a_k :=& \langle f, A_k \rangle, \quad b_k := \langle f, B_k \rangle \end{aligned} \tag{2.8}$$

The division of $a_0$ by two is due to the fact that $||A_0|| = 2$. The series $(a_i)_{i \geq 0}$ and $(b_i)_{i>0}$ are called the *sine-cosine-Fourier Series of* $f$. Using complex numbers we can put the the Fourier Series in equation (2.8) in a more compact form. Euler's identity[4] states

$$\begin{aligned} e^{ix} &= \cos(x) + i \sin(x) \\ e^{-ix} &= \cos(x) - i \sin(x) \end{aligned} \tag{2.9}$$

or rewritten

$$\begin{aligned} cos(x) &= \left(e^{ix} + e^{-ix}\right)/2 \\ sin(x) &= \left(e^{ix} - e^{-ix}\right)/(2i). \end{aligned} \tag{2.10}$$

[4] A proof of Euler's identity: We start with the Taylor expansions of sin, cos and exp.

$$\begin{aligned} sin(x) &= x - x^3/3! + x^5/5! - x^7/7! + x^9/9! - \cdots \\ cos(x) &= 1 - x^2/2! + x^4/4! - x^6/6! + x^8/8! - \cdots \\ e^x &= 1 + x + x^2/2! + x^3/3! + x^4/4! + x^5/5! + \cdots \end{aligned}$$

and thus

$$\begin{aligned} i \cdot sin(x) =& x - ix^3/3! + ix^5/5! - ix^7/7! + \cdots = x + (ix)^3/3! + (ix)^5/5! + (ix)^7/7! + \cdots \\ cos(x) =& 1 - x^2/2! + x^4/4! - x^6/6! + \cdots \quad = 1 + (ix)^2/2! + (ix)^4/4! + (ix)^6/6! + \cdots \\ cos(x) + i \cdot sin(x) =& 1 + (ix) + (ix)^2/2! + (ix)^3/3! + (ix)^4/4! + (ix)^5/5! + \cdots \end{aligned}$$

The last line equals the series $e^{ix}$ above. Hence

$$cos(x) + i \cdot sin(x) = e^{ix}$$

Finally

$$\begin{aligned}
f(x) =& \frac{1}{\pi}\left(\frac{a_0}{2} + \sum_{k=1}^{\infty} a_k cos(kx) + \sum_{k=1}^{\infty} b_k sin(kx)\right) \\
=& \frac{1}{\pi}\left(\frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \frac{e^{ikx} + e^{-ikx}}{2} + b_k \frac{e^{ikx} - e^{-ikx}}{2i}\right) \\
=& \frac{1}{2\pi}\left(a_0 + \sum_{k=1}^{\infty} a_k e^{ikx} + a_k e^{-ikx} \underbrace{+\frac{b_k}{i}}_{-ib_k} e^{ikx} \underbrace{-\frac{b_k}{i}}_{+ib_k} e^{-ikx}\right) \\
=& \frac{1}{2\pi}\left(a_0 + \sum_{k=1}^{\infty} e^{ikx}(a_k - ib_k) + e^{-ikx}(a_k + ib_k)\right) \\
=& \frac{1}{2\pi}\left(a_0 + \sum_{k=-1}^{-\infty} e^{i(-k)x}(a_{-k} - ib_{-k}) + \sum_{k=1}^{\infty} e^{-ikx}(a_k + ib_k)\right) \\
=& \frac{1}{2\pi}\left(a_0 + \sum_{k\neq 0} e^{-ikx} c_k\right) = \frac{1}{2\pi}\sum_{k=-\infty}^{\infty} c_k e^{-ikx}
\end{aligned}$$

where we defined

$$\begin{aligned}
c_k = a_k + ib_k \quad &\text{if} \quad k > 0 \\
c_k = a_k - ib_{-k} \quad &\text{if} \quad k < 0 \\
c_k = a_0 \quad &\text{if} \quad k = 0
\end{aligned}$$

This seemingly absurd definition makes sense, since the $c_k$ equal:

$$c_k = \frac{1}{2\pi}\int_0^{2\pi} f(y)e^{-iky}\,dy$$

which is not hard to check (using (2.9)). Finally we can write the Fourier series in a very compact form.

## 2.2 Fourier Series and Fourier Transform

**Definition 2.22.** The *Fourier Series* of a function $f \in L^2[0, 2\pi)$ is defined as the series

$$(c_k)_{k\in\mathbb{Z}} = \frac{1}{2\pi}\int_0^{2\pi} f(y)e^{-iky}\,dy$$

and $f$ can be written as

$$f(x) = \sum_{k=-\infty}^{\infty} c_k e^{ikx}$$

If we introduce the function $C_k(x) = \frac{1}{2\pi} e^{-ikx}$ we can express the formulas using the inner product

$$f(x) = \sum_{k=-\infty}^{\infty} \langle f, C_k \rangle C_k(x)$$
$$c_k = \langle f, C_k \rangle$$

The $(c_k)_k$ are called *Fourier coefficients*.

**Remark 2.23.** If the function is not defined on $[0, 2\pi)$ but on $[-T/2, T/2)$ then the Fourier Series is

$$f(x) = \sum_{k=-\infty}^{\infty} c_k e^{\frac{2\pi i}{T} kx}$$
$$c_k = \frac{1}{T} \int_{-T/2}^{T/2} f(y) e^{\frac{-2\pi i}{T} ky} \, dy \tag{2.11}$$

**Remark 2.24.** Functions defined on an interval $[a, b)$ can be identified with periodic functions with period $P = b - a$ by repeating it. Periodic means $f(x + P) = f(x)$ for all $x \in \mathbb{R}$. See figure 2.5 for a visual explanation. In the same manner we can identify periodic functions with period $P$ with functions defined on $[0, P)$.

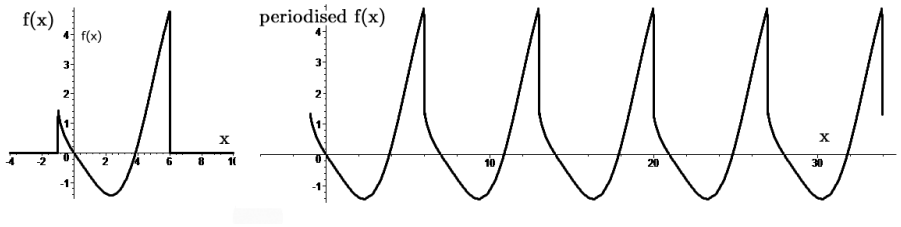



Figure 2.5: Periodising a function by repeating it

**Remark 2.25.** If the function is not periodic (which is usually the case with music[5]) but decays fast enough (e.g. $f \in L^2$), we can say the period of the function is infinite. We compute what that means for our definition of the Fourier Series in equation (2.11) if $T$ would equal $\infty$.[6]

$$f(x) = \sum_{k=-\infty}^{\infty} c_k e^{ikx} = \sum_{k=-\infty}^{\infty} \frac{1}{T} \int_{-T/2}^{T/2} f(y) e^{\frac{-2\pi i}{T} ky} \, dy \, e^{\frac{2\pi i}{T} kx}$$
$$= \sum_{k=-\infty}^{\infty} \frac{1}{T} \int_{-\infty}^{\infty} f(y) e^{-2\pi i \frac{k}{T} y} \, dy \, e^{2\pi i \frac{k}{T} x} \tag{2.12}$$

If we define the integral

$$\hat{f}(\nu) = \int_{-\infty}^{\infty} f(y) e^{-2\pi i \nu y} \, dy$$

[5]Not considering Vexations by Satie.

[6]One cannot do calculations with $\infty$. So one must be careful and this here is quite heuristic.

and set $\nu = k/T$, than (2.12) takes the form of a Riemann sum

$$\sum_{k=-\infty}^{\infty} \frac{1}{T} \hat{f}(\tfrac{k}{T}) e^{2\pi i \frac{k}{T} x}$$

which goes towards the following integral as $T$ goes towards $\infty$.

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

Finally we obtain

**Definition 2.26.** The Fourier transform of a function $f$ ($f \in L^2(\mathbb{R})$) is

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

$\hat{f}$ is called the *Fourier transform of f*. There are also the notations $\hat{f}(y) = (\mathcal{F}f)(y) = (\mathcal{F}f(x))(y) = F(y)$ in use. Note that the two equations only differ in the sign of the exponent. The first is $-2\pi i y x$, the second is $+2\pi i y x$.

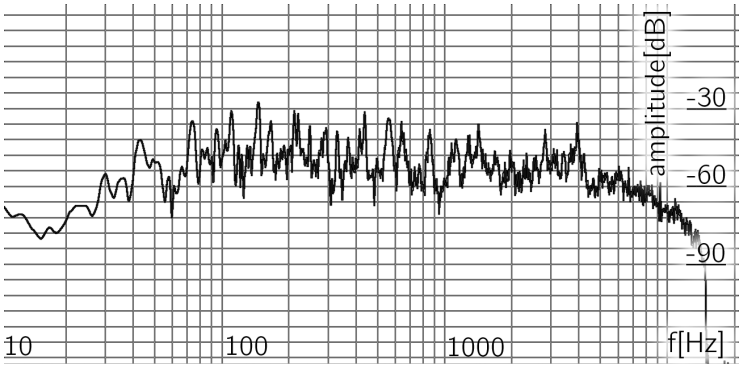



Figure 2.6: The Fourier transform of *Help* by the *Beatles*

The Fourier Series and the Fourier transform are very closely related. The Fourier coefficients of a periodised function equals the value of the Fourier transform of the original function up to a constant. Because a computer cannot compute infinite long integrals, which is necessary in the case of the Fourier transform, one usually computes a kind of Fourier Series on the PC. Because of the strong relationship between these two methods, this is often called "Fourier transform" too.

If we apply the Fourier transform to a whole musical piece we can see the information about the magnitudes of the frequency components of the whole piece easily, though all spatial information is hidden in the phase of the coefficients. Since music[7] is very seldom stationary (in the sense that it widely

[7]Not consideringASLSP by John Cage.

changes over time), it is not of much use to analyse a whole piece at once. See for example figure 2.6 the magnitude Fourier transform of the song *Help* by the *Beatles* is plotted. The x-axis describes the frequencies and the y-axis the amplitudes. In that picture we can only see, which frequencies are used often by the Beatles, but we cannot localize any given note any more. One can say, the Fourier transform somehow averages the over the time.

Therefore it is useful to cut music into small pieces, each only some milliseconds long, and do a Fourier transform on every bit. That way we can see all of the frequencies at certain time intervals. This method is called *Short Time Fourier transformation* which will be discussed later. Beforehand the concept of frames must be introduced.

## 2.3 Frames

In the following I will often speak of *signals*, being be the functions to be analysed and we suppose that they are well behaved (like continuous, integrable or whatever we need else).

ONBs have a lot of useful properties (they are orthonormal, the coefficients in the expansion (2.1) are unique, they are a minimal set which spans the vector space, ...) but they also have disadvantages. If one element of the basis is lost, the remaining set does not span the vector space any more (i.e. there are vectors which cannot be written as a sum of basis vectors). Similar vectors can have very different representations, e.g. noisy signals, spatial translated vectors, et cetera. When using *frames* instead of vectors some of theses disadvantages can be avoided. A frame can be seen as an overcomplete basis. They are robust against input perturbations like quantization effects, can be translation invariant and much more. The main advantage of frames is, that the frame elements (usually called *atoms*) can be adapted to the input signal. E.g, if one has music using a 12 tone scale, one can adapt the atoms to represent the tones of that 12 tone scale. If one wants to analyse speech, one could choose atoms which resemble vowels and consonants. Also unequally spaced sampling is possible. Disadvantages are, the loss of orthogonality of the basis vectors and no uniqueness of the coordinate coefficients in general, which also leads to redundancy (more coefficients needed than in the orthogonal case to represent any given signal). A big disadvantage is, that the frame cannot be used to resynthesise the original signal - instead a *dual frame* must be used and its computation is not trivial. For a long time the only way to compute it was using iterative procedures. In 1990 Wexler and Raz [16] proved a duality condition which allowed to compute the dual frame in the discrete case via solving a system of linear equations. Finally frames are computational more costly.

**Definition 2.27.** Let $I$ be a countable or finite index set. The family of functions $(\varphi_i)_{i\in I}$ over the Hilbert space $\mathcal{H}$ are called a *frame* for $\mathcal{H}$ if there exist constants $0 < A \leq B < \infty$ such that

$$A\,||f||^2 \leq \sum_{i\in I} |\langle f, \varphi_i\rangle|^2 \leq B\,||f||^2 \tag{2.13}$$

If $A = B$ then the frame is called a *tight frame*. In that case one can assume without loss of generality $A = B = 1$.

To retrieve the original vector $f$ one in general needs the *dual frame* denoted with $(\gamma_i)_{i\in I}$. It then holds that (proof in [3, p. 40])

$$f = \sum_{i\in I} \langle f, \varphi_i \rangle \gamma_i \tag{2.14}$$

This sum is called the *signal expansion of* $f$. The coefficients

$$c_i = \langle f, \varphi_i \rangle \tag{2.15}$$

are called the frame *expansion coefficients* or *frame analysis* of f.

**Remark 2.28.**

- If the frame is tight the frame equals its dual frame [3, page 42].
- From equation (2.13) it can be seen that any unitary operation on a frame yields a new frame with the same frame bounds. Unitary operators are bijective linear operators between Hilbert spaces and preserve the inner product, i.e. let $U : \mathcal{H}_1 \to \mathcal{H}_2$ be unitary and $f, g \in \mathcal{H}_1$, then $\langle Uf, Ug \rangle_{\mathcal{H}_2} = \langle f, g \rangle_{\mathcal{H}_1}$
- If $\mathcal{B}$ is an ONB for $\mathcal{H}$ than $\mathcal{B}$ is a frame for $\mathcal{H}$ with $A = B = 1$.

**Example 2.29.** The *Mercedes Frame* (figure 2.7) for $\mathbb{R}^2$. Let $w_1 = (0, 1)$, $w_2 = (\sqrt{3}/2, -1/2)$, $w_3 = (-\sqrt{3}/2, -1/2)$ and $v = (v_1, v_2)$. This an tight frame since for $v \in \mathbb{R}^2$

$$\begin{aligned}\sum_{n=1}^{3} \langle v, w_n \rangle^2 &= (0 \cdot v_1 + v_2)^2 + \left( \frac{\sqrt{3}}{2} v_1 - \frac{1}{2} v_2 \right)^2 + \left( \frac{\sqrt{-3}}{2} v_1 - \frac{1}{2} v_2 \right)^2 \\ &= \frac{3}{2} v_1^2 + \frac{3}{2} v_2^2 = \frac{3}{2} ||v||^2\end{aligned}$$

Hence all $v \in \mathbb{R}^2$ satisfy

$$v = \sum_{n=1}^{3} \langle v, w_n \rangle w_n$$

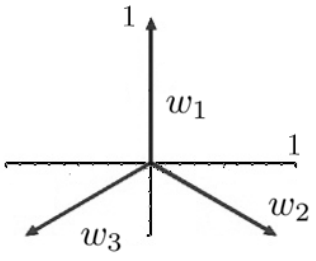



Figure 2.7: The Mercedes frame.

# Chapter 3

# Linear Time-Frequency Representations

## 3.1 Short Time Fourier Transform

The STFT works as follows: We first cut the signal into pieces via multiplication with time shifted window functions. *Window functions* are functions from $\mathbb{R} \to \mathbb{C}$. They usually have the shape of a bump and are symmetric respective the y-axis. We will often use continuous and compactly supported (i.e. zero outside some interval) functions. The easiest window function has the shape of a rectangle (and therefore is called *Rectangular Window*). This function is zero until some point, then it jumps onto 1, and at a second point it jumps back to zero where it stays. Applying this window to a signal is equivalent to cutting away everything before and everything after these two points. After windowing the signal, a Fourier transform of each portion is applied. To resynthesise we inverse Fourier transform and multiply with possibly different time shifted window functions.

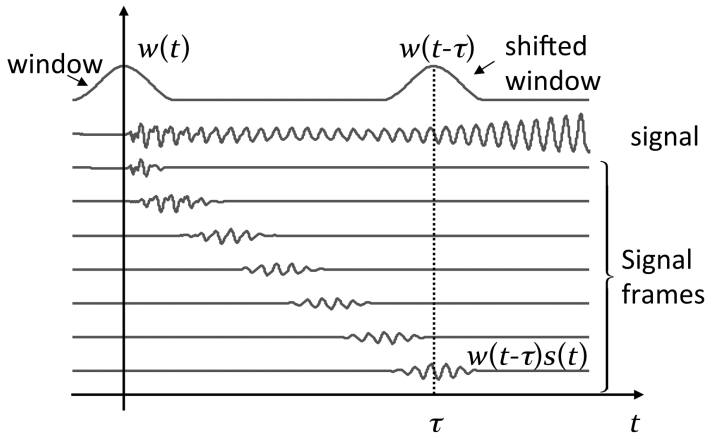



Figure 3.1: Windowing of a signal using a smooth window. Image taken from [8]

Windowing can can introduce problems. First, the window function can induce discontinuities of the cut signal. Second, it introduces frequencies into the signal which are not present, since windowing a function with a window of length $T$ makes this function $T$-periodic when represented by Fourier series.. This effect is called *spectral leakage*. Appropriate choice of the window function can only reduce the leakage. Both problems are depicted in figure 3.2

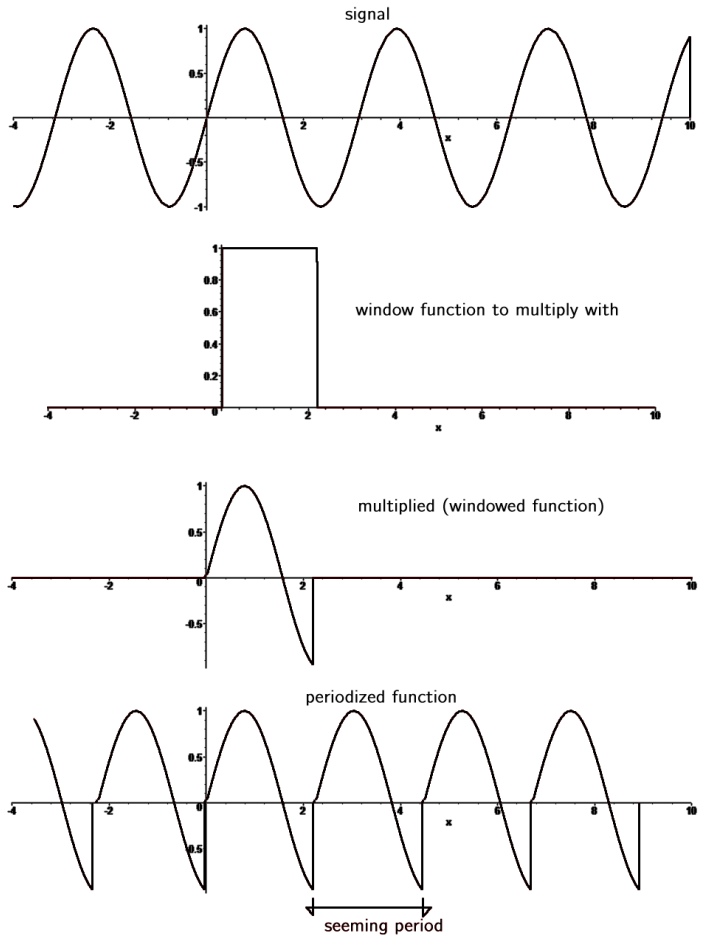



Figure 3.2: Windowing of a signal using the rectangular window.

**Definition 3.1.** Let $s$ be a signal, $h$ a window function. The *(uniform) STFT of $s$ with respect to $h$*, $\mathcal{S} : L^2(\mathbb{R}) \to (\mathbb{R}^2 \to \mathbb{C})$ is obtained by applying the operator $\mathcal{S}$ to the signal $s$:

$$[\mathcal{S}s](\tau, \nu) = \int_{-\infty}^{+\infty} s(t)\overline{h(t-\tau)}e^{-2\pi i\nu(t-\tau)}\,dt \quad \text{with } \tau, \nu \in \mathbb{R}$$

The overbar denotes complex conjugation. The conjugation of the window

function will make notation easier afterwards. Rewriting this leads to

$$[\mathcal{S}s](\tau,\nu) = \int_{-\infty}^{\infty} s(t)e^{-2\pi i\nu(t-\tau)}\overline{h(t-\tau)}\,dt$$

From this notation, one can also interpret the STFT as the the output signal around time $\tau$ of a bandpass filter centred at frequency $\nu$. Windows haves a certain duration in time (in other words they work with the signal on some time interval and not only at one point), but they also have a certain interval of frequencies on which they are operating. This interval of frequencies is called *bandwidth of the window*. To analyse the signal with a good time resolution we need short windows, for a good frequency resolution we need windows with a narrow bandwidth. Unfortunately we cannot have both at the same time. This is due to the uncertainty-principle. Without loss of generality we can suppose $f$ is normalized, i.e. $||f|| = 1$. From Plancherel's theorem it follows that $\hat{f}$ is also normalized. Then the following inequality holds (supposed everything exists):

$$\left(\int_{-\infty}^{\infty} |t\,f(t)|^2\,dt\right)\left(\int_{-\infty}^{\infty} \left|\nu\,\hat{f}(\nu)\right|^2\,d\nu\right) \geq \frac{1}{16\pi^2} \tag{3.1}$$

The first factor denotes the variance of the window in time. The second factor analogously denotes the variance of the Fourier transform of the window. A proof can be found in any harmonic analysis book, for example [14, page 133]

There is a third way to read the STFT's definition. For that, we introduce the following two operators:

**Definition 3.2.** Let $h : \mathbb{R} \to \mathbb{C}$. The time shift operator $\mathbf{T}_\tau$ and the modulation operator $\mathbf{M}_\nu$ $(\tau, \nu \in \mathbb{R})$ are defined as

$$\mathbf{T}_\tau h(t) = h(t-\tau)$$
$$\mathbf{M}_\nu h(t) = h(t)\cdot e^{2\pi i\nu t}$$

Furthermore we will use the abbreviation:

$$h_{\tau,\nu} = \mathbf{T}_\tau \mathbf{M}_\nu h$$

$h_{0,0}$ therefore equals $h$. The modulation operator works as a frequency shift of the window in the frequency domain since $(\mathcal{F}\mathbf{M}_\nu h)(\omega) = (\mathcal{F}h)(\omega-\nu)$. We will further identify the time-frequency shifted window $h_{\tau,\nu}$ with the corresponding point $(\tau,\nu) \in \mathbb{R}^2$ in the TF-plane.

With this notation we can rewrite the STFT as follows

$$[\mathcal{S}s](\tau,\nu) = \int_{-\infty}^{+\infty} s(t)\overline{h(t-\tau)}e^{-2\pi i\nu(t-\tau)}\,dt = \int_{-\infty}^{+\infty} s(t)\overline{h_{\tau,\nu}}\,dt = \langle s, h_{\tau,\nu}\rangle$$

That way we can interpret the STFT as the projection of the signal onto the shifted and modulated windows. These windows can be interpreted as points (actually regions) in the time-frequency plane. We will utilize the third point to generalize the STFT. Because of the importance, I repeat the three possibilities of how to understand the STFT [3]:

- *At a specific time, the STFT is the Fourier transform of the signal multiplied by a local analysis window function.*

- *At a specific frequency, the STFT is essentially the output signal of a bandpass filter centered at the respective frequency.*
- *At a specific point in the TF plane, the STFT is the inner product of the signal with a version of an analysis window TF-shifted to the respective TF point.*

Before going on, I present the song *Help* (this time only the beginning) analysed via the Short Time Fourier transform (STFT). At time $t = 10\,s$ one can easily see the striking *Hee*$^{e\,ee}$*elp* exclamation at the beginning of the song (figure 3.3).

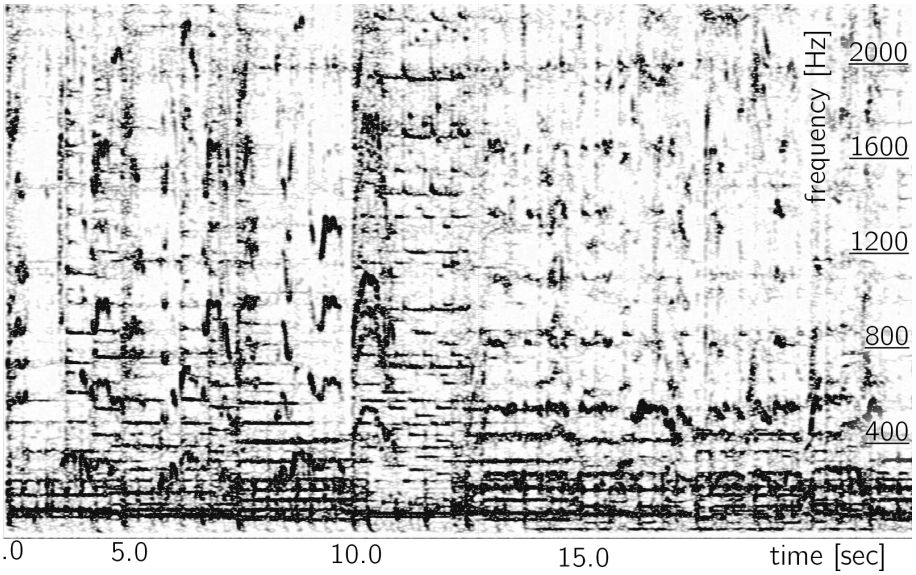



Figure 3.3: A STFT of *Help* by the *Beatles*

## 3.2 Gabor Expansion

The *Gabor Expansion* GE (also called *Gabor transform, Gabor Analysis, Phase Vocoder*) is closely related to the STFT. While in the STFT $\tau$ and $\nu$ for time-frequency shifting the window function are continuous, *they are discrete in the Gabor Expansion. Therefore, the Gabor Expansion provides a partitioning (tiling) of the the TF-plane into rectangles of equal size* [3]. Gabor suggested using the Gaussian bell function for the windows and a so called *critical tiling*, which means that the product of the time shift parameter $a$ and the frequency shift parameter $b$ equals one. Later on, this was generalized to other windows and other lattice densities. Concerning the two parameters, a distinction is made between three types

- $a \cdot b < 1$: oversampling
- $a \cdot b = 1$: critical sampling
- $a \cdot b > 1$: undersampling

Critical sampling results in poor numerical stability but uniqueness of the coefficients. Oversampling yields better stability at the cost of non-unique Gabor coefficients. In the case of undersampling the time-frequency shifted windows are not a frame any more and thus are of no interest as the signal cannot be recovered from them. A proof can be found in [11].

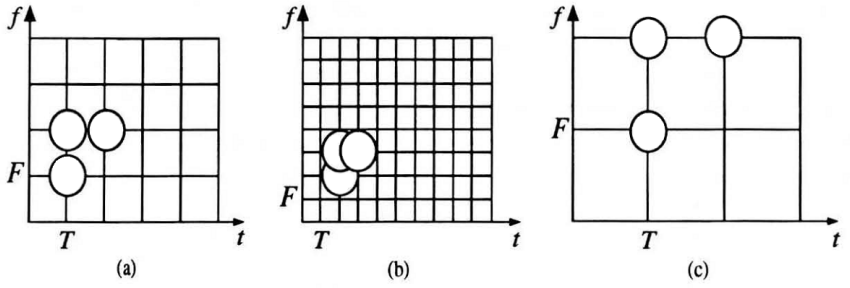



Figure 3.4: Tiling of the time-frequency plane provided by the Gabor expansion. (a) Critical sampling, (b) oversampling, (c) undersampling. Image taken from [3]

The set of functions given by a window function $h$ together with two parameters $a$ and $b$

$$\mathcal{G}(h,a,b) = \{h_{na,qb} | n,q \in \mathbb{Z}\} = \{\mathbf{T}_{na}\mathbf{M}_{qb}h | n,q \in \mathbb{Z}\}$$

is called *Gabor System*. $n$ will be called *time index*, $q$ will be called *frequency index*. A Gabor System is not an orthonormal basis in general. Hence statements about existence and uniqueness of the Gabor coefficients are rather involved. It turned out that the concept of *frames* can answer this questions in a satisfactory manner. A Gabor system that is a frame is called a *Gabor Frame* or *Weyll-Heissenberg Frame*. In the setting of Gabor frames, I will use the abbreviation $h_{n,q} = h_{na,qb}$. The dual frame to a Gabor frame is itself a Gabor frame generated by a dual window $\tilde{h}$. If the window $h$ has a finite length $a \leq l < \infty$, then $\tilde{h}$ is a dual window to $h$ if their product overlap-adds to one, i.e.

$$\sum_{r=-\infty}^{\infty} h(t-rl)\tilde{h}(t-rl) = 1 \quad \forall t \in \mathbb{R}$$

If $h = \tilde{h} = \sqrt{\varphi}$ where $\varphi$ is a function which overlap-adds to one, i.e. $\sum_{r=-\infty}^{\infty} \varphi(t-rl) = 1$ for all $t$, this is clearly fulfilled. Also if one of the windows overlap adds to one, and the other is a rectangular window (with an appropriate window length) this is fulfilled.

In section 3.1 we saw that due to the uncertainty principle it is impossible to have functions with small bandwidth and small time support. In the case of the GE the uncertainty is much bigger. If $\mathcal{G} = \{h_{n,q}\}$ is a Gabor frame such that $ab = 1$, i.e. critical sampling, then

$$\left(\int_{-\infty}^{\infty} |t\,h(t)|^2\,dt\right)\left(\int_{-\infty}^{\infty} \left|\nu\,\hat{h}(\nu)\right|^2\,d\nu\right) = +\infty$$

The statement, named Balian-Low theorem, says if one of the factors is finite, the other must be infinite. A good proof can be found in [2].

Due to that theorem, frames were sought-after which are orthogonal and consist of atoms with finite uncertainty.

## 3.3 Wavelet Transform

Like the GE, the Wavlet transform (WT) provides an expansion of the signal. The TF tiling of the WT is quite different from that of the GE. The WT provides a tiling where lower frequencies are analysed with lower time resolution and higher frequencies with higher time resolution, where the GE divides the TF plane into rectangles of the same size. Figure 3.5 compares the GE and the WT tiling. The WT has some advantages over the GE. It is more natural to our human ear in the sense, that our ear has a coarser frequency resolution for higher frequencies, too. That means, the good frequency resolution of the Gabor Expansion for high frequencies is unnecessary for our ears, on the other hand our ear has a much finer frequency resolution for low frequencies than the STFT usually does.

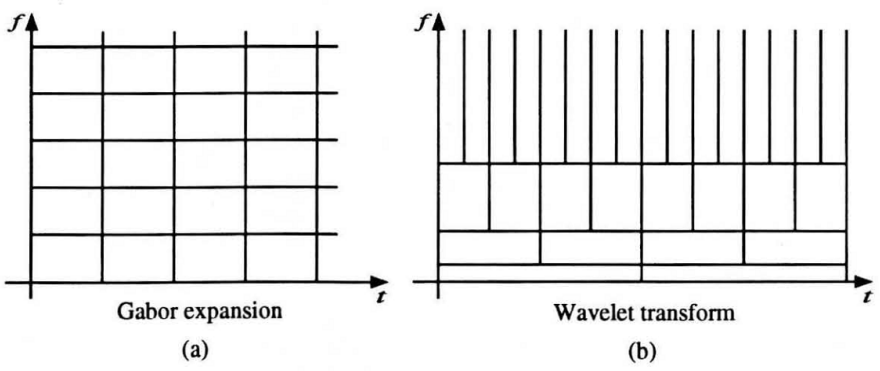



Figure 3.5: Tilings of the time-frequency plane corresponding to GE (a) and WT (b). Image taken from [3].

The frame for the Wavelet transform is generated by shifting and scaling a window function (usually called *mother wavelet*) instead of shifting and modulating as in the GE case. Let $0 \neq \psi \in L^2(\mathbb{R})$ such that

$$\int_{-\infty}^{\infty} \psi(t)\, dt = 0$$

and the admissible condition

$$\int_0^{\infty} \frac{\left|\hat{\psi}(\nu)\right|^2}{\nu}\, d\nu < \infty$$

is fulfilled. Then the family of *wavelet atoms* $\{\psi_{\tau,\lambda} | \tau \in \mathbb{R}, \lambda \in \mathbb{R}^+\}$ is defined as

$$\psi_{\tau,\lambda}(t) = \frac{1}{\lambda}\psi\left(\frac{t-\tau}{\lambda}\right)$$

The *Wavelet transformation (WT)* of $f \in L^2(\mathbb{R})$ with respect to $\psi$ is defined as

$$\mathcal{W}_\psi f(\tau, \lambda) = \langle f, \psi_{\tau,\lambda} \rangle = \frac{1}{\lambda} \int_{-\infty}^{\infty} f(t) \overline{\psi\left(\frac{t-\tau}{\lambda}\right)}\, dt$$

An introduction to wavelet theory can be found in [6] and [1]

Other non-constant tilings of the TF-plane building up on the Wavelet transform were subsequently introduced [5][13][15][17].

## 3.4 Redressed Warped Gabor Expansion

Recently, in [9] a type of Gabor Expansion with warped frames was introduced, which allows for a tiling of the TF plane into arbitrary frequency bands and time intervals. The STFT's uniform bands are transformed into non-uniform bands via a frequency and a time map, i.e. monotonically increasing functions remapping the frequency axis and the time axis. However, since time and frequency are not independent variables, the time mapping influences the frequency as well as the frequency mapping influences the time. This would lead to a skewed tiling of the TF-plane as artistically drawn in figure 3.6 which makes perfect reconstruction of signals more complicated and renders applications for visualizing sound unusable. In the same work it has been shown, that given that the atoms have limited bandwidth, the frequency dependent time delays can be redressed. Nevertheless, windows with limited support in frequency have infinite time support, meaning they are useless for real-time computation. In a subsequent paper Evangelista [7] proposed approximations and ideas for the design of nearly perfect reconstruction analysis and synthesis atoms, which allow for the real-time computation of time-frequency representations on non-uniform frequency bands.

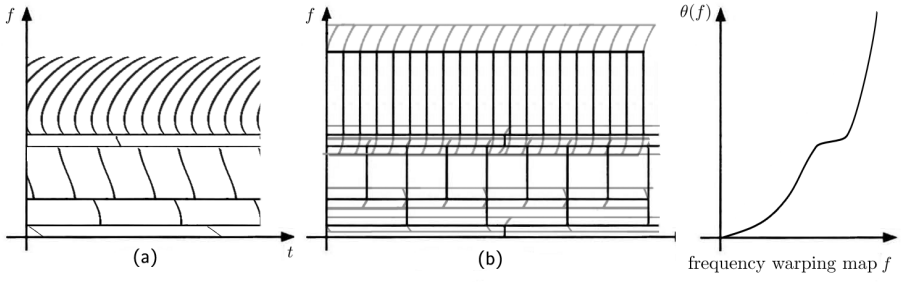



Figure 3.6: Tiling of the time-frequency plane with warped Gabor atoms (a) and redressed Warped Gabor atoms (b) together with the corresponding warping map.

## Warping Gabor Frames

This section summarises the results presented in [7]. Let $\theta : \mathbb{R} \mapsto \mathbb{R}$ be a strictly monotone increasing function, $s$ a signal. The frequency warping operation is then completely characterized by a function composition operator in the frequency domain

$$\hat{s}_w = \hat{s} \circ \theta,$$

If the warping map $\theta$ is one-to-one and a.e. differentiable, then a unitary form of the warping operator can be defined by the frequency domain action

$$\left(\widehat{\mathbf{U}_\theta s}\right)(\nu) = \sqrt{\left|\tfrac{d\theta}{d\nu}\right|}\hat{s}(\theta(\nu))$$

The pre-factor ensures that the energy of the window is preserved under warping and is needed for $\mathbf{U}$ to be a unitary operation. To construct the redressed Gabor system we start from a Gabor frame $\{\varphi_{n,q}\}_{n,q\in\mathbb{Z}}$ with dual frame $\{\gamma_{n,q}\}_{n,q\in\mathbb{Z}}$

$$\varphi_{n,q} = \mathbf{T}_{na}\mathbf{M}_{qb}h$$
$$\gamma_{n,q} = \mathbf{T}_{na}\mathbf{M}_{qb}g$$

where $h$ and $g$ are dual windows. In the application we will use windows which are dual to themselves, i.e. $h = g$ and hence $\varphi = \gamma$. The warped frame is obtained by the following observation:

$$s = \mathbf{U}_\theta^\dagger \sum_{n,q\in\mathbb{Z}} \langle \mathbf{U}_\theta s, \varphi_{n,q}\rangle \varphi_{n,q} = \sum_{n,q\in\mathbb{Z}} \langle s, \mathbf{U}_\theta^\dagger \varphi_{n,q}\rangle \mathbf{U}_\theta^\dagger \varphi_{n,q}$$

$\mathbf{U}_\theta$ is unitary, hence $\mathbf{U}_\theta^\dagger = \mathbf{U}_\theta^{-1} = \mathbf{U}_{\theta^{-1}}$. Defining the warped atoms as

$$\tilde{\varphi}_{n,q} = \mathbf{U}_{\theta^{-1}}\varphi_{n,q}.$$

the expansion (2.14) of the signal $s$ reads

$$s = \sum_{n,q\in\mathbb{Z}} \langle s, \tilde{\varphi}_{n,q}\rangle \tilde{\varphi}_{n,q}.$$

and the Fourier transforms of these warped atoms are

$$\hat{\tilde{\varphi}}_{n,q}(f) = \sqrt{\tfrac{d\theta^{-1}}{df}}\hat{h}(\theta^{-1}(f) - qb)e^{-2\pi i\theta^{-1}(f)na}$$

The time shift factor $e^{-2\pi i\theta^{-1}(f)na}$ is not linear in $f \in \mathbb{R}$ and therefore the atoms bear frequency dispersive delays. Evangelista constructed a further warping operation which linearises the delays in certain bands using the series $(\eta_{m,q})_n$

$$\eta_{m,q}(n) = \int_{-\frac{1}{2}}^{+\frac{1}{2}} \sqrt{\tfrac{d\vartheta_q}{d\nu}} e^{2\pi i(n\nu - m\vartheta_q(\nu))}, \quad \text{with } n, m \in \mathbb{Z},$$

where $\vartheta_q : (-\frac{1}{2}, +\frac{1}{2}) \mapsto (-\frac{1}{2}, +\frac{1}{2})$ is an almost everywhere differentiable onto and one-to-one map. This forms an orthonormal basis [4][12] and thus the mapping $\mathbf{D} : \ell^2(\mathbb{Z}) \to \ell^2(\mathbb{Z})$ defined as

$$\mathbf{D}c(m) = \langle c, \eta_{m,q}\rangle$$

is unitary. The map $\vartheta_q$ can be extended over the entire real axis as congruent modulo 1 to a 1-periodic function. The index $q$ denotes dependence on the band-index. Defining the redressed atoms as

$$\tilde{\tilde{\varphi}}_{n,q} = \sum_m \eta_{n,q} \tilde{\varphi}_{m,q}$$

one obtains the signal expansion

$$s = \sum_{n,q\in\mathbb{Z}} \langle s, \tilde{\tilde{\varphi}}_{n,q}\rangle \tilde{\tilde{\varphi}}_{n,q}. \tag{3.2}$$

and the Fourier transform of the redressed atoms become

$$\hat{\tilde{\tilde{\varphi}}}(f) = \sqrt{\tfrac{d\theta^{-1}}{df}}\,\sqrt{\tfrac{d\vartheta_q}{d\nu}}\Big|_{\nu=a\theta^{-1}(f)} \hat{h}(\theta^{-1}(f)-qb)e^{-2\pi i n\vartheta_q(a\theta^{-1}(f))}. \tag{3.3}$$

Hence, the effect of the dispersive delays would be counteracted if

$$\vartheta_q(a\theta^{-1}(f)) = d_q f \tag{3.4}$$

for any $f \in \mathbb{R}$, where $d_q$ are positive constants controlling the time scale in each frequency band. In this case, the Fourier transforms of the redressed atoms becomes

$$\hat{\tilde{\tilde{\varphi}}}_{n,q}(f) = \sqrt{\tfrac{d_q}{a}}\hat{h}(\theta^{-1}(f)-qb)e^{-2\pi i n d_q f}. \tag{3.5}$$

*However, each map $\vartheta$ is constrained to be congruent modulo 1 to a 1-periodic function, while the global warping map $\theta$ can be arbitrarily selected. Furthermore, having to be one-to-one in each unit interval, the functions $\vartheta$ can at most experience an increment of 1 there.* [7] This means, a perfect redressing is only possible in a certain bounded interval, restricting this approach to atoms with limited bandwidth (the so called *painless case*). Let us first examine the painless case, for simplicity we assume that the bandwidth of $h$ is $Kb$ where $K$ is a positive integer, i.e. $\hat{h}(f) = 0$ for all $|f| \geq Kb/2$. The equation (3.4) therefore must be fulfilled for all $f$ with $\left|\theta^{-1}(f) - qb\right| < Kb/2$ because of (3.3) . In order to do that, we construct the map $\vartheta_q$ as follows. We first periodise the function $\theta$ in the interval $[\nu^-, \nu^+)$, with $\nu^- = -\frac{Kb}{2} - qb$ and $\nu^+ = \frac{Kb}{2} - qb$. Then we scale it so that is has a period of 1 and only experience an increment of 1 in each period.

Rewriting equation (3.4) with $\nu = \theta^{-1}(f)$ yields

$$\vartheta_q(a\nu) = d_q\theta(\nu) \quad \text{for} \quad \theta(-\tfrac{Kb}{2}+qb) < \nu < \theta(\tfrac{Kb}{2}+qb). \tag{3.6}$$

With figure 3.7 it is easy to determine the constant's $a$ and $d_q$ value. In the $\nu$ direction we need to scale it at least by $Kb$, which yields

$$1/a \geq Kb \tag{3.7}$$

In the $\theta(\nu)$ direction we need to scale it at least by the bandwidth of the $q^{th}$ warped window $B_q = \theta(\nu^+) - \theta(\nu^-)$ which yields

$$1/d_q \geq B_q = \theta(\nu^+) - \theta(\nu^-) \tag{3.8}$$

The first condition 3.7 does not depend on $q$ and is satisfied if we assign sufficient oversampling. For the second condition 3.8 we need to choose $d_q \leq 1/B_q$. Also, if there is an upper bound to our bandwidths we can choose identical $d = \sup_q d_q$. If our signal is band-limited we can choose $d = \max_q d_q$ since we only need a finite number of bands. Then, all samples would be aligned to a uniform time scale throughout frequencies. Nevertheless, this is not possible for real-time purposes, since it increases the computational load too much.

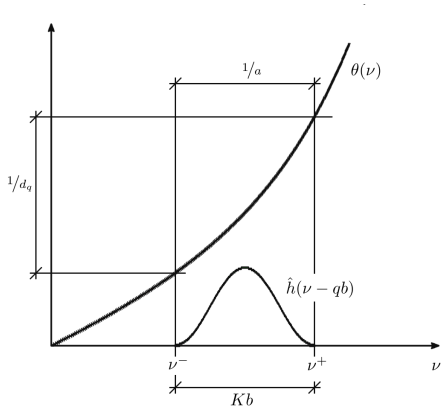



Figure 3.7: The function $\theta$ and how to scale it.

In the general case, a perfect time realignment of the components is not guaranteed. Locally, within essential bandwidths of the warped modulated windows it is possible to linearise the phase of the complex exponentials. The greyed slanted ends of the TF-boxes in figure 3.6 shall visualize this. The black boxes depict the essential bandwidth and length of the atoms, the greyed slanted ends visualize the overlap of the atoms. Anyway, by construction the redressed warped Gabor systems are guaranteed to be frames for any choice of the maps $\vartheta_q$ satisfying the stated periodicity conditions.

## Windows with unbounded support

For real-time applications, through warping another problem emerges. The modulated frequency warped windows will not have compact support in the time domain in general, even if the original windows have had this property. Evangelista discusses two approaches for this problem. The first is to linearise the map $\theta$ around the point $\theta(qb)$ in equation (3.5). This is possible since $\theta$ is differentiable. The warped windows then are

$$\tilde{\tilde{\varphi}}_{n,q}(t) = \theta'(qb)h(\theta'(qb)(t - na))e^{2\pi i\theta(qb)(t-na)}$$

It turns out that this approach does not lead to good reconstruction.

The second method is to compute the windows and to truncate them afterwards to a desired length. Because it is unlikely that we find a closed analytical form of the Fourier-transformed-warped-inverse-Fourier-transformed window, we have to compute it numerically, thus using the Discrete Fourier transform (DFT) for the Fourier transformations. The DFT has the property,

that the size of the input vector is as big as the size of the output vector. Since the output window will have unbounded support, we need a long (ideally infinite) output vector and hence a long input vector by appending zeros on both sides of the input vector.

### Applications of redressed Gabor-Expansion

These warped Gabor frames bear many similarities with Wavelets, hence they share many of their advantages. Later (figure 4.2) we will see that warped Gabor atoms and wavelets also have great similarities in their visual shape.

- The atoms frequency bands can be adapted to the scale of the signal, e.g. twelve tone scale, pentatonic scale, etc. figure 3.8 shows two spectrograms, one adjusted to the scale of the signal and uniform one.
- The attack of a sound is usually much shorter than somebody can play distinct notes. The note duration is in the range of several milliseconds, while virtuoso passages are seldom faster than 800 attacks per minute, respective at a spacing of 75 ms. Hence one can choose a finer time resolution at attacks, resolving it well in time and finer frequency resolution afterwards, making it possible to precisely track glissandi or vibrati.
- The bands can be adapted to partials of the signal. E.g. the piano has non-harmonic partials in the lower register. Also bells and drums have non-harmonic partials. The FFT with its equal spaced frequency bands is not well suited for the analysis of non-harmonic partials.
- From my experience while writing this thesis, this analysis-synthesis algorithm can be used to achieve interesting sound effects and to synthesise new sounds. I tried the following: Using different atoms for analysis and synthesis; reorder the atoms before synthesis; do various calculations with the coefficients; skip certain coefficients.
- Generation of non uniform spectrograms.
- Warped Gabor frames are easier to compute than wavelets.

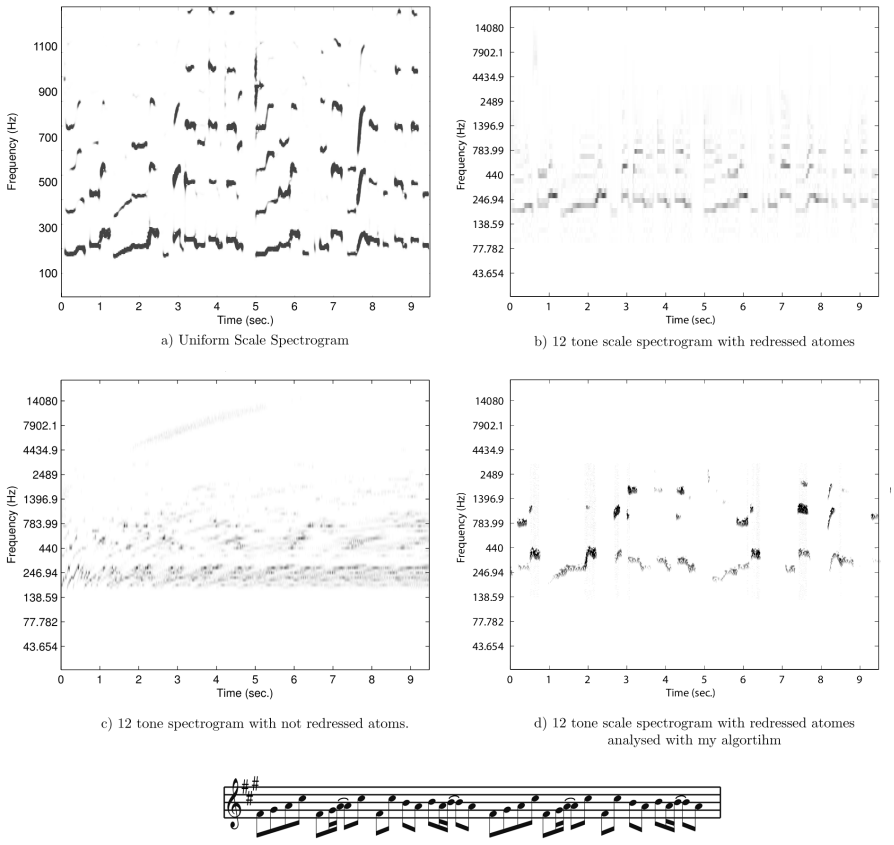



Figure 3.8: Four times the the beginning of *Tom's Diner* by Susanne Vega. The singing phrase is represented in the score line below. Plot a) shows it analysed with uniform frequency bands. Plot b) is a nonuniform 12-tone scale spectrogram showing clearly the melody. Plot c) uses the same nonuniform 12-tone scale but with not redressed atoms. Plot d) is the same as b) but analysed using my algorithm while it was in an early development stage. The plots b), c) and the score line are taken from [9] with permission from Prof. G. Evangelista.

# Chapter 4

# Realtime Computation of Warped Gabor Expansion

## 4.1 Problems due to real time computation

If one does not need real time computation, one can use windows which are unbounded in time. This has the advantage that we can use windows which are compactly supported in the frequency domain (band limited). For real time computation this is obviously not possible. That means, we have to use windows which are unbounded in the frequency domain. This makes problems in the redressing of the Gabor frames which are discussed above.

Let $\mathrm{WL}_q$ denote the length of the window $\tilde{\tilde{\varphi}}_{n,q}$. For computing one coefficient $c_q$ from (2.15) we need to measure the signal $s$ for a duration of $\mathrm{WL}_q$ to compute the inner product. Therefore an algorithm has at least a delay of $\mathrm{WL}_q$. This delay is different for each band. To reconstruct the signal one clearly has to wait $\max_q \mathrm{WL}_q$. The longest window will usually be the one with the lowest center frequency. The delay of the algorithm hence usually depends on the window with the lowest frequency band.

In an implementation of the warped Gabor Expansion, one cannot use the Fast Fourier transform (FFT), because because the center frequencies of the analysis/synthesis bands are not uniformly spaced. The computational costs are therefore much higher than they are in STFT (or Wavelet analysis) applications.

## 4.2 The implementation of the algorithm

I will first present details on how I implemented the algorithm, split up into the parts *Computation of all parameters*, *Window computing*, *Analysis* and *Synthesis*. This part will be mostly independent of my concrete implementation, meaning independent of the programming language, Pure Data, data types and so on. The part *Further details* will deal with this.

### Computation of all parameters

For my implementation I made the following assumptions:

- The warping map $\theta$ is odd, i.e. $\theta(-f) = -\theta(f)$, bijective on $\mathbb{R}$, strictly monotone increasing and differentiable.
- The window function $h$ is the raised cosine window given by

$$h(t) = \begin{cases} \sqrt{\frac{2b}{R}}\, cos\frac{\pi t}{\mathrm{WL}_h} & \text{if } \frac{-\mathrm{WL}_h}{2} < t < \frac{\mathrm{WL}_h}{2} \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

  where $\mathrm{WL}_h$ is the window length, $b$ is the frequency sampling parameter, $R$ is an integer and we set the time sampling parameter

$$a = \mathrm{WL}_h/R \tag{4.2}$$

  Note that this function is real valued and dual to itself since $h^2$ is the Hanning window.
- The input signal is real valued.

To compute the time-frequency sampling parameters, I followed the ansatz in [7]. $R$ is chosen after the essential bandwidth BW in which one wants to linearise the phase. In figure 4.1 one can see the magnitude of the Fourier transformed cosine window. The main lobe has bandwidth $3/\mathrm{WL}_h$. Linearising inside this bandwidth leads to $R \geq 3$ using (4.2) and (3.7). Setting $\mathrm{WL}_h = \frac{R}{2f_0}$ makes sense in the case of a tempered scale warping map in order to have sufficient frequency resolution. If $f_0$ is the frequency of the smallest tone to be represented, then adjacent tones fall away from the main frequency lobe of the window. This gives

$$a = \frac{\mathrm{WL}_h}{R} \tag{4.3}$$

Since the warped windows have infinite support in general, I compute the windows using window length $\mathrm{WL}_c$ ($c$ for compute), which I defined as

$$\mathrm{WL}_c = \frac{\mathrm{WL}_h}{\theta'_{\mathrm{inf}}}\, C_{wl}$$

where $C_{wl} \geq 1$ can be chosen inside the PD-patch and $\theta'_{\mathrm{inf}} = \inf_{f\in\mathbb{R}} \theta'(f)$. If $\theta'_{\mathrm{inf}} = 0$, I set $\mathrm{WL}_c = 5\, C\, \mathrm{WL}_h$. $C_{wl} \approx 3$ gives good results. Since $ab \leq 1/R$ in order to obtain a frame, I define

$$b = \frac{1}{aRC_b}$$

where $C_b \geq 1$ can be chosen inside the PD-patch. In my tests $C_b = 2$ provided good results.

For further calculations I defined

$$B_q = \theta(qb + \frac{\mathrm{BW}}{2}) - \theta(qb - \frac{\mathrm{BW}}{2})$$

which will be called *essential bandwidth* of the $\mathrm{q}^{th}$ window. The parameters $d_q$ must obey $d_q \leq 1/B_q$. Hence I defined them as

$$d_q = \frac{1}{B_q C_d}$$

where $C_d \geq 1$ can be chosen inside the PD-patch. $C_d \approx 2,5$ yielded good results. Too small and too big numbers both gave worst results. Remark that the numbers $d_q$ have to be chosen such that $d_q/\mathrm{SR} \in \mathbb{N}$, where SR is the sampling rate, because in the applications we have to deal with a sampled version of the signal $s$. Since this will not be the case in general, one has to choose smaller numbers.

Finally the number of bands $q_{sup}$ we need for a given SR is

$$q_{sup} = floor(\theta^{-1}(SR/2)/b) + C_q,$$

The extra term $C_q$ is due to the widening of the bands after warping. Empirical tests showed that it is enough to compute bands up to a center frequency of about 34 kHz. Clearly, this depends on the specific parameters used, especially the warping map and the overlap in the frequency domain and should be adjusted after a change of parameters.

Actually, all window functions which are real valued and dual to itself can be used. In that case only the computation of the parameters changes. Particularly, the window functions need not to be symmetric.

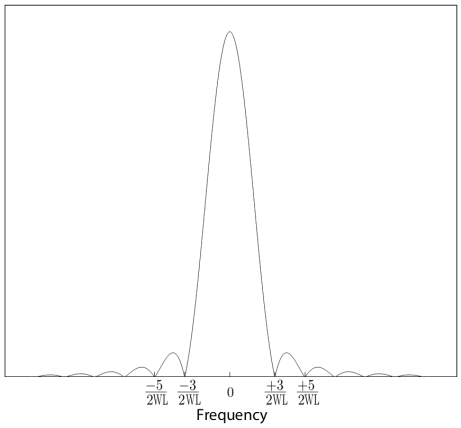



Figure 4.1: Magnitude Fourier transform of the cosine window. The image is taken from [7].

## Window Computing

Using some simplifications and observations one can reduce the computational load considerably. Since for the painless case the windows $\tilde{\tilde{\varphi}}_{n,q}$ are time shift invariant under a shift of $d_q$ (see equation (3.5))

$$\tilde{\tilde{\varphi}}_{n,q}(t) = \tilde{\tilde{\varphi}}_{0,q}(t - nd_q)$$

it suffices to compute the window $\tilde{\tilde{\varphi}}_{0,q} = \mathcal{F}^{-1}\sqrt{d_q/a}\hat{h}(\theta^{-1}(f) - qb)$ and shift it in time. In the following I will omit the time index $n$ in $\tilde{\tilde{\varphi}}_{n,q}$. In the general case the delay is not linear outside the band of interest. For real time computation we need to precompute the windows, hence we have to make the approximation that the windows still are time shift invariant for the general case. In the not

redressed setting, the windows are totally not time-shift invariant, precluding the precomputation due to memory restrictions. In my applications all windows taken together needed about 30 MB. However, if one uses a weakly rising warping function and big overlap in frequency the required space easily takes several GB leading to problems for 32-bit architecture PCs.

Using an odd function for the warping map, one only needs to compute half of the windows since then $d_{-q} = d_q$ and

$$\begin{aligned}\hat{\tilde{\varphi}}_{-q}(-f) &= \sqrt{\tfrac{d_{-q}}{a}}\hat{h}(\theta^{-1}(-f)+qb) = \sqrt{\tfrac{d_q}{a}}\hat{h}(-(\theta^{-1}(f)-qb)) \\ &= \overline{\sqrt{\tfrac{d_q}{a}}\hat{h}(\theta^{-1}(f)-qb)} \quad \text{because } h \text{ is real valued} \\ &= \overline{\hat{\tilde{\varphi}}_q(f)}\end{aligned}$$

Applying the inverse Fourier transform we get

$$\tilde{\tilde{\varphi}}_{-q}(t) = \overline{\tilde{\tilde{\varphi}}_q(t)}$$

and finally since our signal is real valued ($s = \bar{s}$)

$$c_{-q} = \langle s, \tilde{\tilde{\varphi}}_{-q}\rangle = \langle s, \overline{\tilde{\tilde{\varphi}}_q}\rangle = \int_{-\infty}^{\infty} s(t)\tilde{\tilde{\varphi}}_q(t)\,dt = \int_{-\infty}^{\infty} \overline{s(t)}\tilde{\tilde{\varphi}}_q(t)\,dt = \overline{\langle s, \tilde{\tilde{\varphi}}_q\rangle} = \overline{c_q}.$$

In my implementation I used the raised cosine window as the window function $h$. Since its Fourier transform has a closed expression in mathematical terms

$$\hat{h}(f) = \sqrt{\frac{b}{2R}}\left(\operatorname{sinc}\left(\frac{\mathrm{WL}_h - 1}{2}\right) + \operatorname{sinc}\left(\frac{\mathrm{WL}_h + 1}{2}\right)\right) \tag{4.4}$$

I can use this formula directly in the computation afterwards.

## Windows to be computed

Tests (not included in here) indicated that a precomputation of the windows with a higher sampling rate yields better results up to factor of $C_{SR} = 4$. Using double precision and long double precision numbers (using gcc 4.8.1 under MinGW, $\varepsilon = 10^{-19}$) does not seem to make a difference, other than using single precision which results in an analysis-synthesis error 7 dB higher (for white noise).

### Pseudo Code

I present the pseudo code to calculate the windows. To compute the inverse FFT I used the fftw-library. This library expects the frequency 0 at bin 0.

```
SR=44100;
N_out=WL_c*SR;
N=N_out/C_SR;                    //I assume this is an integer.
win_c=array of size q_sup*N;
win=array of size q_sup*N_out;   //array for the warped window.
deltaF=1/WL_c;                   //step size of FFT.
```

```
I2F(i):=i*deltaF;               //index to Frequency.
for(q=0 to q_sup-1){
  for(i=0 to  N/2){             //positive frequencies.
    win_c[q][i]= h_hat(theta_inv(I2F(i))-q*b)*sqrt(d_q/a);
  }
  for(i=iN/2+1 to i<N-1){       //negative frequencies.
    win_c[q][i]=h_hat(theta_inv(I2F(i-iN))-q*b)*sqrt(d_q/a);
  }
  i0=SR/(2.0*deltaF);            //Set all frequencies higher
  for(i=i0 to N-i0){             //than SR/2 to zero.
    win_c[q][i]=0;
  }
  pac=fft_inverse(pac);         //Compute inverse FFT

  for(int i=0 to N-1){          //Normalisation due to the FFT
    win_c[q][i]=win_c[i]*deltaF;
  }
  //Dependent on the used fft-implementation other orderings of
  //the frequencies and normalisations in win_c[q] may be needed
  for(i=0 to N_out-1){          //down sampling.
    win[q][i]=win_c[i*C_SR];
  }
}
```

### Functions used in the algorithm

**h_hat** is the Fourier transformed of $h$ from equation (4.1) and is given in equation (4.4). **theta_inv** is the inverse of the frequency warping map $\theta$.

### Pictures of precomputed windows

In figure 4.2 some windows can be seen. The parameters for these windows are:

- R = 3
- WL = 0,125 s
- $\mathrm{WL}_c$ = 0,5625 s
- a = 0,041667 s, b = 4 Hz
- C_SR=1
- Oversampling: 2
- $h$: raised cosine window
- $\theta(\nu) = \begin{cases} \frac{2f_0\nu}{d} & \text{if} \quad |\nu| \leq 2d \\ \sigma(\nu) f_0 2^{\frac{|\nu|}{d}} & \text{otherwise} \end{cases}$
  with $d = 36$, $f_0 = 12$ and $\sigma(\nu)$ denotes the sign of $\nu$

The first few windows look like usual TF-shifted Gabor frames, since the warping map $\theta$ is linear for small frequencies. When the warping map grows exponential the windows look like Wavelets. They grow in amplitude while their support shrinks.

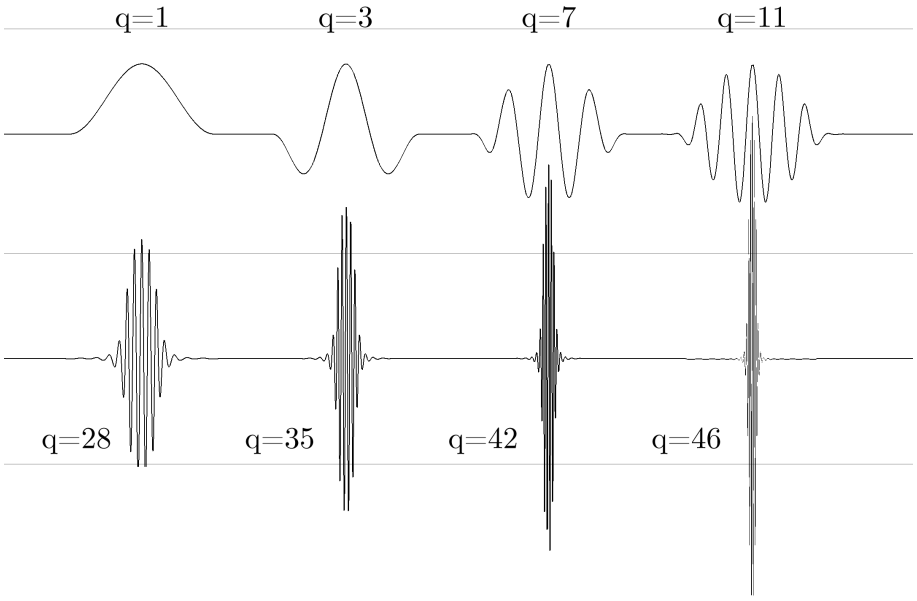



Figure 4.2: Some windows $\tilde{\tilde{\varphi}}_q$.

## Window Cutting

Due to the warping, the supports of the windows are in general unbounded even if they were bounded before. It is therefore indispensable to cut the windows after warping. In my algorithm I implemented two different ways to cut the windows.

- For the first approach I use the fact that scaling a function in time by the factor $a$ leads to a Fourier-transformed which is scaled by the inverse, $f(ax) \overset{\mathcal{F}}{\longmapsto} 1/a\hat{f}(\omega/a)$. If one linearises $\theta'$ around the point $\theta(qb)$ as done in [7, equation (37)], the warped windows become

$$\tilde{\tilde{\varphi}}_q(t) = \theta'(qb)\, h(\theta'(qb)t)\, e^{2\pi i \theta(qb)t} \tag{4.5}$$

  If $\mathrm{WL}_h$ denotes the length of the original window $h$ we get:

$$\mathrm{WL}_q = \frac{\mathrm{WL}_h}{\theta'(qb)}$$

  This calculations also show that one may need a larger window length for computing warped windows, because they could get stretched in time if $\theta' < 1$.

- Since the Fourier transform decays to zero for functions in $L^2$ it makes sense to set the windows at the beginning and the end equal to zero until the point where the amplitude is higher than a given number.

Both techniques led to very similar cutting conditions. Since it turned out that the second approach does not work well with windows whose center-frequency is higher than the Nyquist frequency, I only used the first technique to cut the windows. It is important that the windows are cut with respect to the numbers $d_q$. Otherwise they would be wrongly aligned in time.

**Pseudo Code**

```
for(q=0 to q_sup-1){
  WL_new=WL_c/d_theta(q*b)*C_WL;   //compute new window length.
  if(WL_new>WL_max) WL_new=WL_max; //compare with maximum length.
  t0=(WL_c-WL_new)/2;          //difference to current length in s.
  N0=t0*PD_SAMPLING_RATE;      //sample to begin with.
  N1=N-N0;                     //last sample.
  if((N-N1) % d_q != 0) N1=N- d_q + (N-N1) % d_q;
                               //round down to multiple of d_q.
  N_new=N1-N0;                 //new length in samples.
  if( N_new >= N) {            //if length is longer do nothing.
    N_new=N; //Nichts tun
  } else {
    win[q]=array of size N_new;    //array for the new window..
    memmove(from: &win[q][N0], to: win[q][0], N_new many values);
                               //move window inside array
                               //new window length = N_new.
  }
}
```

`d_theta` is the first derivative of the frequency warping map $\theta$. `C_WL` is factor greater equal one which can be chosen inside the PD patch. `WL_max` is the maximum window length in seconds, selectable inside the PD patch.

## Analysis

The analysis part poses two difficulties. First we have to buffer the input signal. Let `input` be the input signal passed to the algorithm in blocks with block size `BS`. Second we have to shift the windows $\tilde{\tilde{\varphi}}_q$ in time respective to $d_q$. For this purpose I introduced a counter variable `counter` which counts all samples through. If *counter* mod $d_q = 0$ then a coefficient $c_q$ must be computed. Since *modulo* is a costly operation this part of the program could be optimized further.

Let `snd` be the array for the sound buffer. `snd_A` the pointer to the begin of the sound buffer (sound stArt), `snd_E` the pointer to one after the end of the sound buffer (sound End), `snd_C` the current pointer the sound buffer where we have to copy new parts to (sound Current). The sound buffer is longer than the longest window `win_q` and a multiple of the block size `BS` in which we process the signal. `win_A` the pointer to the begin of `win[q]` and `win_R` a pointer to `win[q]` (window Read), `WL_q` the length of the array `win[q]`. `c` be a struct which contains all the important information needed to reconstruct the signal. The struct contains the starting time `c.start`, the band number `c.band` and the value of $\langle s, \tilde{\tilde{\varphi}}_{n,q} \rangle$ `c.val`.

### Pseudo Code

I will use the macro `READ` in the listing below

```
READ:= sum_r+= (*snd_R) * real(*win_R); //compute inner product.
       sum_i-= (*snd_R) * imag(*win_R);
       --snd_R; --win_R;                //regress the pointers.
```

Now the listing of the Analysis part

```
//Copy input signal into buffer
snd_C=snd_C+BS;     //advance the pointer by BS
if(snd_C==snd_E){  //if we are at the end of the sound buffer
    snd_C=snd_A;    //we go back to the beginning.
}
copy(from: *(input[0]), to: snd_C, BS many values);

//Compute the coefficients
for(i=0 to BS){
  ++counter;         //Increase the counter by one.
  for(q = 0 to q_sup-1){
    if(counter%d_q==0){
      snd_R=snd_C+i;         //where to start reading.
      snd_RA=snd_R+1-WL_q;   //where to end reading,
                             //I am processing backwards here.
      win_A=*(win[q][0]);    //Pointer to the beginning of win[q].
      win_R=win_A+WL_q-1;    //where to start reading.
      sum_r=0;               //real part of the coefficient.
      sum_i=0;               //imaginary part of the coefficient.
      if(snd_RA>=snd_A){     //win[q] fits in the sound buffer,
                             //see figure 4.3 for details.
        while(win_R>=win_A){ READ }
      } else {        //win[q] protrudes the sound buffer.
        while(snd_R>=snd_A){ READ }
        snd_R=snd_E-1;
        while(win_R>=win_A){ READ }
      }
      val_r=sum_r/SR; //The inner product is an integral, hence
      val_i=sum_i/SR; //we have to multiply with the step width.
      c.start=i;
      c.val=sum_r+I*sum_i; //I is the imaginary unit.
      c.band=q;
    }
  }
}
```

## Synthesis

The synthesis part is slightly more complicated in aligning everything right since we have to shift everything forward in time with respect to the starting time

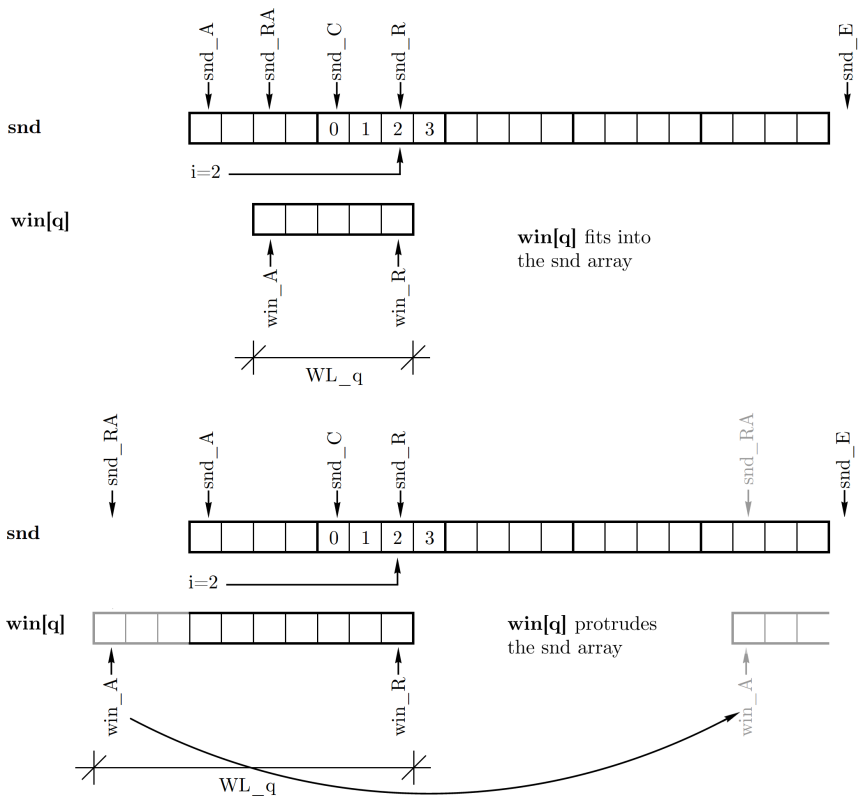



Figure 4.3: The two possible alignments of the **win[q]** array relative to the **snd** buffer. The thick framed, four cells long parts depict the blocks in which we process the signal. If the window win[q] protrudes the sound buffer, we have to shift the projecting parts to the end of the sound buffer.

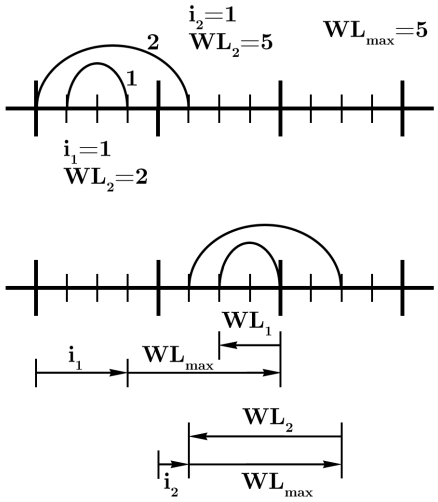



Figure 4.4: Drawn are two windows 1 and 2. Window 1 ends in the first block with starting time $i_1$=3, the second ends in the second block with starting time $i_2$=1. From the begin of the block we have to shift each window for $i_q + \mathrm{WL}_{max} - \mathrm{WL}_q$, resulting in a time delay of $\mathrm{WL}_{max}$ in total.

`c.time` and with respect to the longest window `win_q`. From figure 4.4 it is easy to see how to shift the windows.

Let `snd` be the sound buffer for the output. `snd_A` is pointer to the beginning of the buffer, `snd_W` a pointer to the place where we want to write, `snd_WA` the pointer to the place where we want to start writing to. `snd_WE` the pointer one after the place where we want to stop writing, `snd_E` the pointer to one after the last element of `snd`. `snd_L` will be the length of the sound buffer, this number is a multiple of the block size `BS` and longer than the longest window `win[q]`. The length of the longest window is `WL_max`. `win_A` is a pointer to the begin of `win[q]`, `win_E` a pointer to one after the last entry, `win_R` a pointer where we want to read values from, `win_L` the length of the window `win[q]` The output will be written to `out`. `c=c[0]..c[num_coeff-1]` will be an array of the above mentioned structs, containing all computed coefficients of the current signal block.

### Pseudo Code

I will use the macro `WRITE` in the listing below

```
WRITE:= *snd_W=*snd_W + real(c[k].val) * real(*win_R)
           - imag(c[k].val) * imag(*win_R);   //I only need to
        ++snd_W;++win_R;                      //compute the real part.
```

Now the listing of the Synthesis part

```
k=num_coeff;
while(k){
  --k;
  q=c[k].band;
```

```
  if(q=0) c[k].val=c[k].val*0.5; // I only computed half of the
    //  coefficients since all coefficients but the zero-th
    //  occur two times in the sum.
  snd_WA=snd_C+WL_max+c[k]-WL; //see figure 4.4 for details
  snd_WE=snd_WA+WL;
  win_R=win_A;
                                //see figure 4.5 for details.
  if(snd_WE<=snd_E){            //first case in figure 4.5.
    snd_W=snd_WA;
    while(win_R<win_E) { WRITE }
  } else if (snd_WA>=snd_E) {   //second case in figure 4.5.
    snd_W=snd_WA-snd_L;
    while(win_R<win_R) { WRITE }
  } else {                      //third case in figure 4.5.
    snd_W=snd_WA;
    while(snd_W<snd_E) { WRITE }
    snd_W=snd_A;
    while(win_R<win_E) { WRITE }
  }
}
//copy buffer to output
for(i=0 to BS) {
  output[i]=2*snd_C[i;           //times two, see above.
  snd_C[i]=0;                    //set sound buffer to zero here.
}
snd_C=snd_C+BS; //advance snd_C
if(snd_C==snd_E) snd_C=snd_A;   //if we are at the end, go back.
```

## Optimisations

### Multi-threading

It is very easy to compute the analysis part in several threads, since we do not need to write data and therefore will not have race conditions. The synthesis part needs some considerations about how to parallelise it.

**Analysis** Every thread gets a part of the bands coefficient to compute. In my case, if I have four threads and 20 bands, then thread one computes the coefficients for band 0 to 4, thread two computes the coefficients for band 5 to 9, and so on. A more intelligent partitioning, but not implemented yet, would be: Thread one computes the coefficients for band 0,4,8,12,16. Thread two handles band 1,5,9,13,17, and so on.

**Synthesis** For the synthesis part I tried two approaches how to parallelise it.

- Thread one computes all the values snd[k] in the sound buffer with $k$ mod $4 = 1$, thread two computes the values with $k$ mod $4 = 2$, et cetera.
- Thread one processes all coefficients from band $q = 0..q_{sup}/4 - 1$, thread two processes all coefficients from band $q = q_{sup}/4..2q_{sup}/4 - 1$, et cetera.

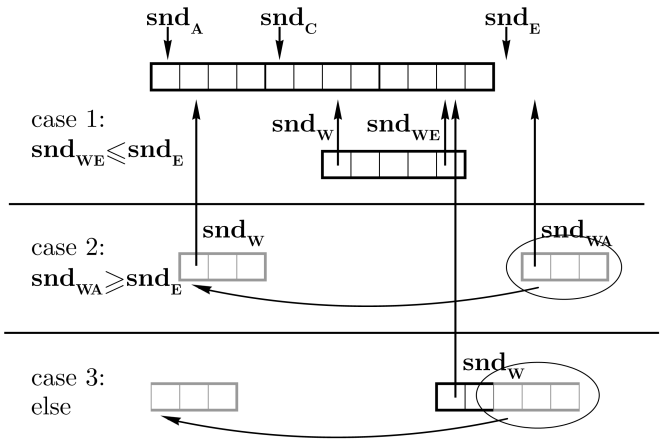



Figure 4.5: The three possible alignments of the **win[q]** array relative to the **snd[]** buffer. The thick framed, four cells long parts depict the blocks in which we process the signal. If the window win[q] stocked out of the sound buffer, we have to shift the projecting parts to the end of the sound buffer. If the window was completely outside, we would have to move everything. The three cases refer to the pseudo code of the synthesis algorithm.

> But each process writes its output into its own sound buffer. When all threads finished work, the main thread sums up the sound buffer. This is not costly, since the main thread only needs to treat the next block size many cells, resulting in an overhead of *Number of threads* additions per sample, which is negligible.

It turns out, that the first approach carries too much overhead (at least in my implementation) and is slower than the single threaded version. Hence I am using the second one.

### SIMD

If one uses single precision numbers (and a new x86 processors) the use of SIMD SSE2 instruction is possible. SSE2 only processes single precision values. This is not a problem since the analysis-synthesis error is the same using single precision and double precision number within the precision of measurement. For double precision vector extensions AVX instructions can be used, but these are not standardized yet. SSE2 instructions take an array of four floating point values which must be 16 bytes aligned. The aligning of the sound buffer is no problem, as long as the block size is a multiple of 16. In my case, I hard coded a block size of 64, which is the standard block size of Pure Data. The aligning of the windows to compute the inner products poses problems.

One solution would be to round the numbers $d_q$ down to a multiple of 4 and let all windows be a multiple of four long. Floating point numbers are 4 bytes long, leading to the result that all windows start at an 16-byte aligned memory location and at one. This increases the computational load slightly

(since all windows are longer and the values $d_q$ are smaller, on the other side the analysis-synthesis error decreases), but is not possible for windows which have a sampling value $d_q$ smaller than 4. This sometimes affects bands with a center frequency from 10 kHz, but usually from 20 kHz onwards. The needed memory stays the same.

Another solution is to make four copies of the windows. The first copy starts 0-bytes-aligned, the second 4-bytes-aligned, etc. Then, for all window functions and for all sample times there is a window function which is 16-bytes aligned. This requires extra computations in order to know which window must be used. These are *Number of bands*×4 reading operations per sample (Actually, we must compute the address modulo 4 resulting in 4 modulo operations, but this reduces to the problem of reading the two lowest bits of the address, due to the binary system). However, because the end of the window will not be 16 byte aligned, one has to add zeros at the end and at the beginning in order to have four floating point values to multiply with the signal. Figure 4.6 explains this graphically. My implementation uses the second approach.

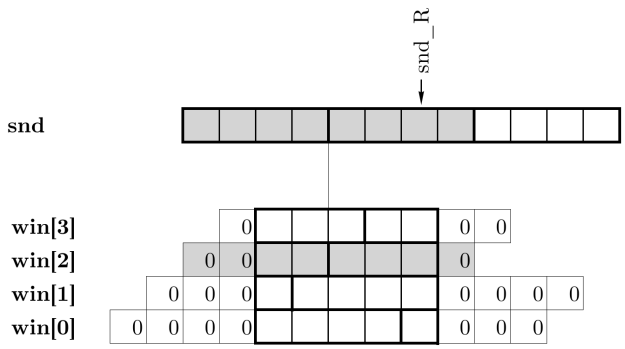



Figure 4.6: The pointer **snd_R** determines the end of the window. The four windows **win[0]** to **win[3]** are all copies of the same window but aligned differently. **win[i]** starts with an i-byte offset. **win[2]** with length 5 has the same alignment as the sound buffer **snd**. The trailing zeros allow it, to compute the inner product of the shaded cells which can be executed with SSE instructions.

### GPU

The algorithm should be perfectly suitable for computation using a GPU. The windows can get precomputed from the CPU with high precision. The windows then are stored in the GPUs memory. Later on, the only data to be transported to and from the graphic card is the sound data.

### Fixed Point Arithmetic

Using fixed point numbers should result in a faster algorithm but accumulated numerical errors can lead to large overall inaccuracy.

## Further Implementation Details

The program is written for a plug-in (External) for Pure Data mainly written by Miller Puckette. It is written in C, version C11, due to the use of the complex data type (C99) and the use of threads (C11). It is compiled using MinGW 4.3.11 under Win7 64 bit. The external is compiled for 32 bit. The code should be easily portable, except for the part with threading, which uses Windows API's. The use of SSE instruction poses no problems since one can use wrapped intrinsic functions provided by Intel. Porting to 64 bit will be problematic in the current version, since I use ugly pointer to int to float conversions. I assume that single precision floats and pointers are 32 bit wide both.

The externals are split up into two major parts. These are the analysis object, called *wabor~* (Warped Gabor), and the synthesis object called *invwabor~* (Inverse Warped Gabor). The objects are connected via two signal connections. Since the data rate of the analysis part is not uniform in time, the signal connection is actually not suitable to transfer the coefficients. Therfore, I use the signal connection to transfer pointers to the actual data. The leftmost connection holds the pointers to the coefficients data (called *Gabor Data* in the following), the second holds the pointers to the windows (called *Control Data* in the following.

### Gabor Data

One audio block $B$ contains the following (and therefore the block size must be at least 20):

`B[0]=0x57414247`: This is just a magic number, "WABG" in ASCII code.

`B[20]`: A pointer to an array of pointers to an array of structs containg the Gabor Coefficients.

The pointer points to an array `c` of size equal to the number of threads used in the analysis part. The number of threads, and therefore the size of the array is stored in the field `tc[WABOR_T_LENGTH_BIN]`. `WABOR_T_LENGTH_BIN` is a macro defined in the header file *wabor_t_definitions.h* and is currently *-5*. Note that this is a negative index. The structs `c[i][j]` with i=0..*number of threads* have the following format:

`int c[i][0].numCoeff`: The number of coefficients stored in c[i]. That means the index variable `j=0 to numCoeff-1`. This field is unused in all other structs contained in c[i].

`int c[i][j].start`: The starting time of the window, same as in the listing above

`float/double c[i][j].val_r`: The real part of the coefficient. Both data types are possible, selectable at compile time.

`float/double c[i][j].val_i`: The imaginary part of the coefficient. The data type is determined at compilation time. Differently than in the listing above I stored the real and imaginary part of the windows in two distinct arrays.

`int c[i][j].band`: The band number q.

### Control Data

One block $C$ contains the following data:

`B[0]=0x57414243`: This is just a magic number, "WABC" in ASCII code.

`B[4]`: Contains the length of the longest window $WL_{max}$ in samples as an integer.

`B[8]`: Contains the number of bands $q_{sup}$ as an int.

`B[12]`: Pointer `paf_r` to an array of pointers to an array containing the real part of the warped windows.

`B[16]`: Pointer `paf_i` to an array of pointers to an array containing the imaginary part of the warped windows.

The data for the windows is organized as follows. The pointer `paf_r` (and respective the pointer `paf_i`) points to an array of size `q_sup`. `paf_r[q]` with $q$ between 0 and $q_{sup} - 1$, contains the warped window. The length of the window $WL_q$ is stored in `paf_r[q][WABOR_T_LENGTH_BIN]`, the window data is stored in `paf_r[q][0]..  paf_r[q][WL-1]`. Before and after the window data there are four trailing zeros. The number $\mathrm{WL}_{max}$ is used in the synthesis part for allocating a big enough sound buffer.

If the synthesis part does not find the Magic Numbers in both the Gabor Data Block and the Control Data Block, then it does nothing. Thus, wrong wirings in the PD patch should not cause a breakdown of the PC.

## 4.3 Computational Costs

### Analysis

A rough estimation yields the following. Let $\mathrm{WL}_q$ denote the length of the $q^{th}$-window in samples. To compute the inner product of that window with the signal one needs $2\mathrm{WL}_q$ many real multiplications and summations. This has to be done every $d_q$ samples. Hence, per sample we have $4\mathrm{WL}_q/d_q$ many operations in average. $\mathrm{WL}_q$ is proportional to $\mathrm{WL}_0/\theta'(qb)$, $d_q$ is proportional to $1/\theta'(qb)$. Summing up over all windows we get the average number of operations per sample $N$:

$$N = \sum_q \frac{4\mathrm{WL}_q}{d_q} \sim \sum_q \frac{4\mathrm{WL}_0}{\theta'(qb)} \frac{\theta'(qb)}{1} = 4q_{sup}\mathrm{WL}_0$$

$q_{sup}$ denotes the number of bands which depends on $\theta^{-1}$.

If we linearise $\theta$ around $qb$, $\theta(qb + \frac{Kb}{2}) \simeq \theta(qb) + \theta'(qb)\frac{Kb}{2}$, we get the estimation for the $d_q$'s.

$$\begin{aligned} B_q &= \theta(qb + \frac{Kb}{2}) - \theta(qb - \frac{Kb}{2}) \simeq \theta'(qb)Kb \\ \Rightarrow d_q &= \frac{1}{B_q} \simeq \frac{1}{\theta'(qb)Kb} \end{aligned} \tag{4.6}$$

Summing everything up, the complexity of the analysis part is proportional to $\mathrm{WL}_0$, K, b and $q_{sup} \sim \theta^{-1}(SR)$.

This is an estimation of the average computational cost. In the worst case all inner products of the windows with the signal have to be computed starting at one frame. The number of operation for that frame is

$$\sum_q 4\mathrm{WL}_q \simeq 4\sum_q \frac{\mathrm{WL}_0}{\theta'(qb)}$$

In our case $\theta(f) \simeq f_0 2^{f/d}$, where $f_0, d \in \mathbb{R}$ are constants with $d > b$ this yields:

$$\begin{aligned}\sum_{q=0}^{q_{sup}-1} 4\frac{\mathrm{WL}_0}{\theta'(qb)} &\simeq \frac{4d\,\mathrm{WL}_0}{f_0 \log 2} \sum_{q=0}^{q_{sup}-1} 2^{-qb/d} \\ &= C_1 \frac{d\,\mathrm{WL}_0}{f_0} \frac{2^{-bq_{sup}/d} - 1}{2^{-b/d} - 1} \\ &\simeq C_1 \frac{d\,\mathrm{WL}_0}{f_0} \frac{1}{1 - 2^{-b/d}}\end{aligned}$$

However, if one processes the audio stream block wise, the worst case cannot arise for all samples in the block at once. Because there cannot be two worst case scenario frames in brief succession (they have a distance of $\max_q d_q$), the next $m$ frames have for sure lesser computational cost (zero if all $d_q > m$). Therefore the average costs are a suitable measure for the analysis process.

### Synthesis Part

The complexity of the Synthesis part is the same as that of the analysis part. Furthermore, in the synthesis part the worst case scenario can be avoided, because only the parts of the next frame buffer have to be computed in real-time, the rest can be computed later. Nevertheless, my algorithm is not optimized in this direction.

### Memory Costs

My algorithm for precomputing the windows needs $2Kq\mathrm{WL}_0$ cells, where $K$ is the oversampling factor used while computing the windows. With slight modifications it should only need $K(\mathrm{WL}_0 + \sum_q \mathrm{WL}_q)$. The smallest possible number of cells being needed is $\sum_q \mathrm{WL}_q$. The analysis and synthesis algorithm both need at least a buffer of size *Audiobuffer-Size* + $\mathrm{WL}_{max}$, where $\mathrm{WL}_{max}$ denotes the window length of the longest windows used in the analysis and synthesis, and *audio buffer-size* the bufferlength in which the audio is processed.

## 4.4 Computational Error

A mathematical estimation of the error would go far beyond this master thesis work. Therefore, I only undertook measurements of the relative error. The error numbers *err* are the difference between the level of the output signal in dB and the level of the input signal in dB (i.e. negative signal to noise ratio). My implementation of the algorithm for real time computation yields values of *err* ≃-50 dB.[1]

### Overlap in Time

The first measurements regards the influence of the windows overlap in time. Overlap 3 means, that the basic window would overlap by one third of it's

[1] For comparison: 16 bit quantization (which is CD-quality) has *err* ≃-96 dB, 8 bit quantization has *err* ≃-50 dB

length.[2] I first performed measurements with white noise to see if this parameter has an influence at all, and if so, in which range the parameter shall be chosen. The results can be seen in figure 4.7. It turned out, the overlap has no big influence on the error. Overlap less than 1,5 yields to undersampling, i.e. the windows do not constitute a frame any more and are of no interest. Overlap factors from 3 to 15 nearly yield an error of about -30 dB. With even higher overlap noise with high frequencies emerges, probably due to numerical instabilities in the algorithm. When using double precision numbers for the synthesis sound buffer the high noise still emerges. If the input signal has no high frequencies, one can low-pass filter the output after the synthesis. In the table in figure 4.7 the values in the column *err after Filtering* denote the err after low-pass filtering with a roll of frequency adapted to the emerging high noise. A test with an input signal with frequencies up to 10 kHz and an overlap factor of 30 gave an error of -63 dB.

As a result of this preliminary test, further tests with different kinds of signals were made with overlap values 3,6, 12 and 18. The number $err_{20k}$ in the tables denotes the error after low pass filtering the output with a biquad filter with roll of frequency 20 kHz. The following signals were used for testing

- **white:** White noise.
- **pink:** Pink noise.
- **brown:** Brownian noise.
- **sine440:** A pure 440 Hz sine tone.
- **sine20k:** A pure 20 kHz sine tone.
- **square440:** A square wave with 440 Hz.
- **square20k:** A square wave with 20 kHz.
- **tri_harm:** A sum of triangular waves with frequencies 440 Hz, 880 Hz, 1320 Hz, 1760 Hz and 2200 Hz (the first five harmonics)
- **const:** A constant signal.
- **clicks:** Clicks with a spacing of 1 s.
- **china:** The piece 杜十娘 (dù shí niáng). A chinese folk song featuring sharp attacks of cymbals, drums and voice, total length 9 s.
- **toms:** The a capella beginning of Susan Vegas song *Toms Diner*, total length 9 s.
- **led:** The beginning of Led Zeppelin's *Kashmir* featuring distorted guitars, total length 17 s.
- **beet:** The beginning of Piano Sonata op 31.2 by Beethoven featuring me, total length 42 s.
- **chopin:** The beginning of Chopin's Ballade No. 4. featuring sustained piano chords, total length 30 s.
- **freq:** A long test file consisting of 24 pure sinusoids (each long 2 s) with following frequencies: 200, 300, 400, 500, 600, 700, 800, 900, 1k, 2k, 3k, 4k, 5k, 6k, 7k, 8k, 9k, 10k, 12k, 14k, 16k, 18k, 20k and 22k Hz. Total length: 23 s.
- **test:** A long test file consisting of different sections. It begins with three types noise (each 2 s), followed by 4 sinusoids with overtones (harmonic and nonharmonic) (each 2 s), followed by one linear sweep and one logarithmic sweep (each 3 s), followed by DTMF tones in different lengths (in

[2] Due to the warping, the warped windows overlap by slightly different values for each band dependent on the overlap of the basic window and the values $d_q$ which depend on the warping map.

| Overlap | err | Overlap | err | err after Filtering |
|---|---|---|---|---|
| 0,003 | -0,0 | 6 | -30,2 | |
| 0,03 | -0,3 | 7,5 | -29,9 | |
| 0,3 | -2,5 | 9 | -29,5 | |
| 1 | -8,2 | 12 | -29,3 | |
| 1,5 | -14,4 | 15 | -26,9 | -48,0 |
| 2 | -22,6 | 18 | -20,0 | -52,9 |
| 3 | -28,2 | 24 | -12,4 | -40,4 |
| 4 | -29,4 | 30 | -8,8 | -41,1 |
| 4,5 | -29,6 | | | |

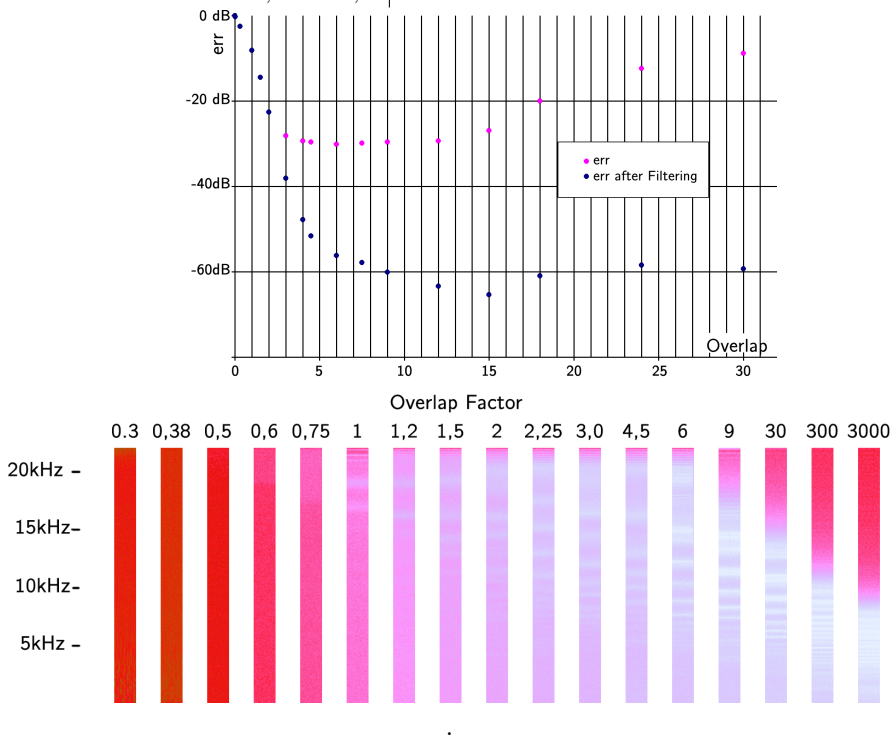



Figure 4.7: Error subject to the overlap of the windows in the time domain. The red values (Overlap $\leq$ 1,5) denote Gabor systems which do not constitute a frame. The columns (made with the program *Cool Edit* show the spectrograms of the difference between the analysed-synthesised and the original signal which is white noise. The intensity is from white (less) over pink to red (very much). Values in dB.

| | |
|---|---|
| EssBw | 24 Hz |
| R | 3 |
| q | 0...101 |
| WL | 0,125 s |
| $WL_c$ | 0,75 s |
| $L_{max}$ | 33075 |
| a | 0,04166 s |
| b | 4 Hz |
| Oversampling | 2 |
| d | 36 |
| $f_0$ | 12 |
| cutfactor | 16,6 |

| $\sum WL_q/d_q$ | | | 4295 | | 8912 | | 18044 | | 26537 | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\sum WL_q$ | | | 1515486 | | 1514912 | | 1512707 | | 1512431 | |
| $\sum WL_q/N$ | | | 14857 | | 14845 | | 14830 | | 14827 | |
| Overlap | | 3 | | 6 | | 12 | | 18 | | |
| Name | err | $err_{20k}$ | err | $err_{20k}$ | err | $err_{20k}$ | err | $err_{20k}$ |
| white | -28,1 | -32,3 | -30,0 | -45,2 | -30,2 | -54,1 | -50,2 | -54,1 |
| pink | -30,8 | -31,3 | -38,3 | -42,6 | -31,4 | -51,0 | -31,4 | -51,0 |
| brown | -40,7 | -40,7 | -48,7 | -48,8 | -58,6 | -59,6 | -58,6 | -59,6 |
| sine440 | -30,1 | -30,1 | -41,1 | -41,1 | -45,0 | -44,0 | -45,0 | -44,0 |
| sine20k | -34,1 | X | -51,9 | X | -54,4 | X | -54,4 | X |
| square440 | -30,1 | -30,1 | -40,9 | -51,2 | -47,0 | -48,3 | -47,0 | -48,3 |
| square20k | -33,1 | X | -40,6 | X | -40,9 | X | -40,9 | X |
| tri_harm | -30,3 | -30,3 | -41,3 | -41,2 | -48,3 | -48,4 | -48,3 | -48,4 |
| const | -51,8 | -51,1 | -52,4 | -52,4 | -61,5 | -61,7 | -61,5 | -61,7 |
| clicks | -25,0 | | -25,4 | -48,2 | -26,5 | -57,1 | -26,5 | -57,1 |
| china | -27,1 | -26,6 | -39,2 | -39,2 | -49,2 | -49,2 | -49,2 | -49,2 |
| toms | -26,9 | -11,6 | -41,2 | -41,2 | -47,0 | -47,0 | -47,0 | -47,0 |
| led | -26,5 | -20,1 | -41,1 | -41,1 | -47,2 | -47,2 | -47,2 | -47,2 |
| beet | -26,7 | -17,0 | -40,9 | -40,9 | -48,5 | -48,5 | -48,5 | -48,5 |
| chopin | -26,8 | -12,9 | -41,0 | -41,0 | -47,2 | -47,2 | -47,2 | -47,2 |

Figure 4.8: Error subject to parameter *overlap*. Values above -30 dB and below below -50 dB are shaded. The table on the left states all parameters used for the measurements. Values in dB.

total 6,5 s) and modulated sinusoids (2 s). The end is white noise faded in and out. Total length: 34 s.

- **wa:** A long test file with a frequency modulated sinusoid with unequal spaced overtones starting at 400 Hz and dropping slowly down to 200 Hz. The modulation frequency at the beginning is 0,5 Hz with a frequency hub of 200 Hz and at the end 5 Hz with a frequency hub of 20 Hz. Total length: 29 s.

All test files and noise were created with the program *Cool edit v2.1*. For the test files no *err* number determined, but a spectrogram was made (Gaussian window, 4096 bands).

Since the *err* values of overlap 12 and 18 were nearly same, no spectrogram for overlap 18 was made. The measuring results are in figure 4.8, the spectrograms are in the appendix on page 60.

## Overlap in Frequency

It turned out that a higher overlap of the windows in the frequency domain with constant overlap in the time domain (constant parameter $a$) has no influence on the error (within the error of measurement). Hence I did no further testing using the test files. The results can be seen in figure 4.9.

## Window Cutting

The next question regards the influence of the window cutting onto the output. First measurements with noise showed an inverse proportional correlation (inside the interval of interest) between the number $\Sigma_q \, WL_q/N$ ($N$ denotes the number of windows) and the error, $4000 \cdot N$ samples correlate to 5 dB.

Tests with warped windows with the linearised warping map showed that the *err* is in the same magnitude as long as the window length is shorter than the window length of the linearised warped window. Clearly the error does not get smaller if one chooses a bigger window length with linearised warped windows.

| b | err |
|---|---|
| 8 | -25,7 |
| 4 | -24,9 |
| 2 | -25,7 |
| 1,6 | -26,0 |
| 1,33 | -26,6 |
| 1,14 | -26,0 |
| 1 | -26,3 |
| 0,89 | -26,2 |
| 0,8 | -25,5 |

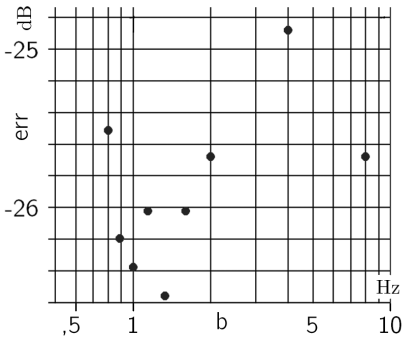



Figure 4.9: Error subject to the parameter $b$, parameter $a = 0,4166\,s$ constant, values in dB. Input signal: white noise.

| err | $\sum_q \mathrm{WL}_q/N$ | err | $\sum_q \mathrm{WL}_q/N$ |
|---|---|---|---|
| -31,6 | 16538 | -27,2 | 7609 |
| -31,1 | 12644 | -26,4 | 7070 |
| -30,3 | 11375 | -25,6 | 6309 |
| -30,4 | 10623 | -25,1 | 5788 |
| -29,9 | 9326 | -22,9 | 5032 |
| -28,9 | 8907 | -21,6 | 4185 |
| -28,8 | 8368 | -19,2 | 2376 |

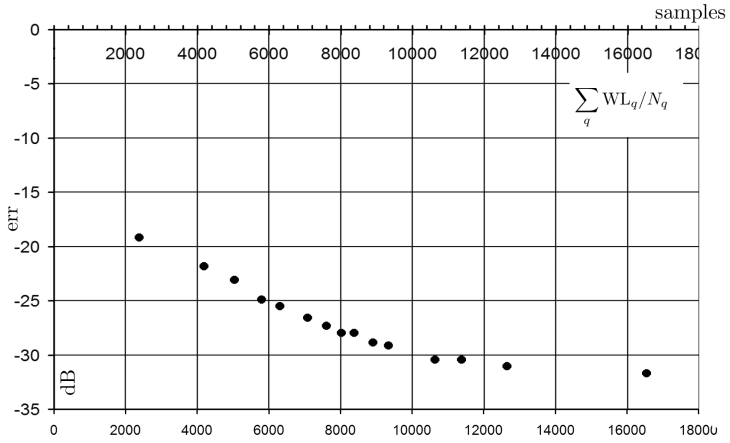



Figure 4.10: Error subject to $\sum_q \mathrm{WL}_q/N$.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| EssBw | 24 | $\sum WL_q/d_q$ | | 1503 | | 4353 | | 12235 | | 30884 | 98897 |
| R | 3 | $\sum WL_q$ | | 262854 | | 720779 | | 1096885 | | 1447696 | 4047784 |
| q | 0...101 | $\sum WL_q/N$ | | 2577 | | 7066 | | 10753 | | 14193 | 39684 |
| WL | 0,125 s | $L_{max}$ | | 9413 | | 24806 | | 24806 | | 24806 | 66150 |
| a | 0,04166 s | $WL_c$ | | 0,5625s | | 0,5625s | | 0,5625s | | 0,5625s | 1,5s |
| b | 4 Hz | cutfactor | | 1,11 | | 3,33 | | 11,1 | | 33,3 | 111 |
| Oversampling | 2 | Name | err | $err_{20k}$ | err | $err_{20k}$ | err | $err_{20k}$ | err | $err_{20k}$ | err | $err_{20k}$ |
| d | 36 | white | -49,5 | -30,9 | -23,7 | -39,2 | -28,5 | -51,5 | -32,7 | -53,3 | -37,7 | -53,6 |
| $f_0$ | 12 | pink | -29,6 | -39,3 | -33,8 | -46,6 | -38,0 | -50,6 | -42,3 | -50,7 | -44,8 | -51,3 |
| Overlap | 12 | brown | -49,8 | -51,3 | -55,3 | -58,4 | -57,9 | -59,3 | -58,6 | -59,4 | -60,1 | -60,0 |
| | | sine440 | -45,7 | -45,7 | -48,1 | -48,1 | -48,0 | -48,0 | -48,5 | -48,0 | -48,1 | -48,1 |
| | | sine20k | -37,3 | X | -23,1 | X | -44,5 | X | -49,0 | X | -54,0 | X |
| | | square440 | -36,0 | -43,6 | -41,7 | -47,6 | -46,2 | -48,3 | -47,5 | -48,0 | -47,8 | -48,0 |
| | | square20k | -33,2 | X | -24,0 | X | -38,9 | X | -46,4 | X | -53,6 | X |
| | | tri_harm | -45,2 | -45,5 | -48,1 | -48,3 | -48,5 | -48,7 | -48,3 | -48,3 | -48,8 | -48,8 |
| | | const | -59,0 | -59,0 | -65,1 | -65,1 | -65,2 | -65,2 | -65,2 | -65,2 | -65,1 | -65,1 |
| | | clicks | -19,5 | -30,9 | -23,6 | -39,1 | -28,5 | -51,4 | -33,4 | -52,6 | -37,4 | -52,3 |
| | | china | -46,4 | -46,4 | -49,3 | -49,3 | -49,9 | -49,9 | -49,4 | -49,4 | -48,0 | -48,0 |
| | | toms | -40,1 | -40,1 | -49,0 | -49,0 | -46,4 | -46,4 | -48,9 | -48,9 | -48,7 | -48,7 |
| | | led | -46,3 | -46,3 | -48,7 | -48,7 | -48,2 | -48,2 | -48,1 | -48,1 | -48,8 | -48,8 |
| | | beet | -46,0 | -46,0 | -48,5 | -48,5 | -49,0 | -49,0 | -48,8 | -48,8 | -49,1 | -49,1 |
| | | chopin | -45,7 | -45,7 | -48,7 | -48,7 | -48,7 | -48,7 | -48,7 | -48,7 | -49,0 | -49,0 |

Figure 4.11: Error subject to the window length. Values in dB.

This shows that the exactly calculated windows with cutting afterwards yields better results than the linearised warped windows.

Due to the big influence of the window length on the error I conducted tests with the test files. The results are shown in figure 4.11, the spectrograms can be seen in the appendix on page 66. These test results too show an inverse proportional correlation between $\sum_q WL_q/N$ and *err* of about $4000 \cdot N$ samples : 5 dB, however, the error does not get smaller than roughly -60 dB.

## Window length for computing

Since the windows get unbounded due to the warping, it is indispensable to calculate the windows with a much longer window than the one of the basic window. This is the topic of the following test. Evangelista used for his tests a window length of about 2,6 s. My tests showed now significant improvement between windows with length 0,75 s and 3 s. The test results are shown in figure 4.12, the spectrograms are in the appendix on page 63.

## Floating Point Precision

There was no difference in the error when used double and float values for the computation of the Gabor coefficients as well as for the synthesis sound buffer.

## Error for unwarped windows

For unwarped windows I got perfect reconstruction. The measurement results are in figure 4.13, the spectrograms are in the appendix on page 69.

| | |
|---|---|
| EssBw | 24 Hz |
| R | 3 |
| q | 0...101 |
| WL | 0,125 s |
| $WL_c$ | 0,75 s |
| $L_{max}$ | 33075 |
| a | 0,04166 s |
| b | 4 Hz |
| Oversampling | 2 |
| d | 36 |
| $f_0$ | 12 |
| cutfactor | 16,6 |

| $\sum WL_q/d_q$ | | 4295 | | 8912 | | 18044 | | 26537 |
|---|---|---|---|---|---|---|---|---|
| $\sum WL_q$ | | 1515486 | | 1514912 | | 1512707 | | 1512431 |
| $\sum WL_q/N$ | | 14857 | | 14845 | | 14830 | | 14827 |
| Overlap | 3 | | 6 | | 12 | | 18 | |
| Name | err | $err_{20k}$ | err | $err_{20k}$ | err | $err_{20k}$ | err | $err_{20k}$ |
| white | -28,1 | -32,3 | -30,0 | -45,2 | -30,2 | -54,1 | -50,2 | -54,1 |
| pink | -30,8 | -31,3 | -38,3 | -42,6 | -31,4 | -51,0 | -31,4 | -51,0 |
| brown | -40,7 | -40,7 | -48,7 | -48,8 | -58,6 | -59,6 | -58,6 | -59,6 |
| sine440 | -30,1 | -30,1 | -41,1 | -41,1 | -45,0 | -44,0 | -45,0 | -44,0 |
| sine20k | -34,1 | X | -51,9 | X | -54,4 | X | -54,4 | X |
| square440 | -30,1 | -30,1 | -40,9 | -51,2 | -47,0 | -48,3 | -47,0 | -48,3 |
| square20k | -33,1 | X | -40,6 | X | -40,9 | X | -40,9 | X |
| tri_harm | -30,3 | -30,3 | -41,3 | -41,2 | -48,3 | -48,4 | -48,3 | -48,4 |
| const | -51,8 | -51,1 | -52,4 | -52,4 | -61,5 | -61,7 | -61,5 | -61,7 |
| clicks | -25,0 | | -25,4 | -48,2 | -26,5 | -57,1 | -26,5 | -57,1 |
| china | -27,1 | -26,6 | -39,2 | -39,2 | -49,2 | -49,2 | -49,2 | -49,2 |
| toms | -26,9 | -11,6 | -41,2 | -41,2 | -47,0 | -47,0 | -47,0 | -47,0 |
| led | -26,5 | -20,1 | -41,1 | -41,1 | -47,2 | -47,2 | -47,2 | -47,2 |
| beet | -26,7 | -17,0 | -40,9 | -40,9 | -48,5 | -48,5 | -48,5 | -48,5 |
| chopin | -26,8 | -12,9 | -41,0 | -41,0 | -47,2 | -47,2 | -47,2 | -47,2 |

Figure 4.12: Error subject to the window length used for computing the windows. Values in dB.

| | |
|---|---|
| EssBw | 44,05 Hz |
| R | 3 |
| q | 0...3000 |
| WL | 0,068027 s |
| $WL_c$ | 0,068027 s |
| a | 0,022676 s |
| b | 7,35 Hz |
| Oversampling | 2 |
| Overlap | 3 |
| $L_{max}$ | 3000 |
| $\sum WL_q/d_q$ | 9021 |
| $\sum WL_q$ | 9021000 |
| $\sum WL_q/N$ | 3000 |

| Name | err | $err_{20k}$ |
|---|---|---|
| white | -44,4 | -87,4 |
| pink | -53,4 | -84,7 |
| brown | -78,9 | -86,9 |
| sine440 | -90,9 | -90,9 |
| sine20k | -93,6 | |
| square440 | -69,9 | -96,6 |
| square20k | -85,6 | |
| tri_harm | -84,9 | -84,9 |
| const | -93,9 | -93,3 |
| clicks | -44,3 | -53,3 |
| china | -79,7 | -79,7 |
| toms | -74,1 | -74,1 |
| led | -86,7 | -86,7 |
| beet | -75,3 | -75,3 |
| chopin | -68,3 | -68,3 |

Figure 4.13: Error for unwarped windows. Values in dB.

# Chapter 5

# Recapitulation

Warped Gabor frames have the following properties

- Easy to adapt to properties of the signals.
- Easy to compute.
- Have infinite time support in general, hence cannot be used for real time computation in the first place without approximations.
- The algorithm in the following condition yields a relative analysis-synthesis error of about -50 dB for typical sound signals and a window length of 0,4 s.

Subjects for further research from very important to negligible.

- The precomputed windows still have numerical errors near $t = 0$. Probably due to this, the *err* numbers are always higher than -60 dB.
- Exact estimation of the error.
- Windows with infinite length could be implemented. Since the windows have infinite support after warping anyway, one is not restricted to use windows with compact time support.
- The external uses a hard coded block size of 64 and a sample rate of 44100 Hz. These values can be changed easily before compilation inside the source code. Change of these values inside PD will crash the external.
- The Bark scale should be implemented as a warping map.
- Further externals should be programmed which
  - add wabor frames together, multiply wabor frames, ....
  - filter all small coefficients out of the wabor frame.
  - generate wabor frames.
- The algorithm requires an analytic version of the Fourier transformed window function. This is an unnecessary restriction.
- The precomputation of the warped windows should get optimised. A lot of operations are performed multiple times, also the memory management is not efficient.

- Tests about the minimum error while still computing in real time should be made. My PC is too old to get meaningful results for that (and suffered some heat strokes while measuring).
- The distribution of the computational load should be better split up between the several threads.
- The external needs a lot of memory for storing meta data which is actually not needed for the analysis-synthesis part.
- The multi-threaded version uses hard coded four threads.
- SIMD instructions are not implemented in the synthesis part yet.

# Chapter 6

# Appendix

## 6.1 Spectrograms

Spectrograms in the following order:

- Spectrograms of the analysed-synthesised-signal **freq, test** and **wa** with varying overlap of the windows in the time domain.
- Spectrograms of the analysed-synthesised-signal **freq, test** and **wa** with varying lengths of the windows used for the precomputation of the windows.
- Spectrograms of the analysed-synthesised-signal **freq, test** and **wa** with varying lengths of the windows used in the analysis and synthesis part.
- Spectrograms of the original signal **freq, test** and **wa** and of the difference between the analysed-synthesised signal and the original signal. The windows used here are not warped, hence are classic STFT windows. Since the error is very small, the spectrograms of the difference signal is amplified by 48 dB, respective 96 dB.

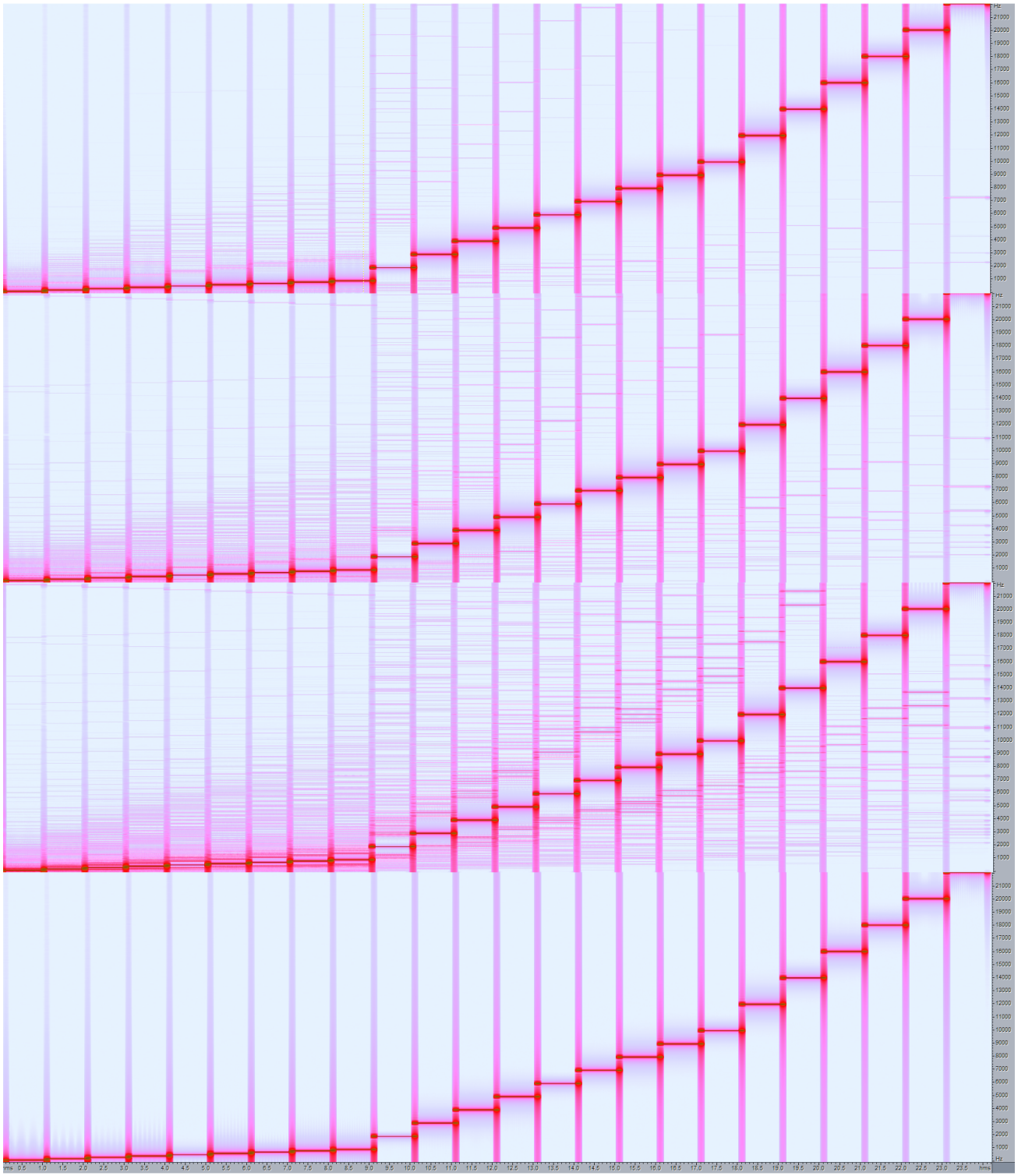
All spectrograms are made with the program *Cool Edit v2.1* using a STFT with Gauss window and 4096 bands.

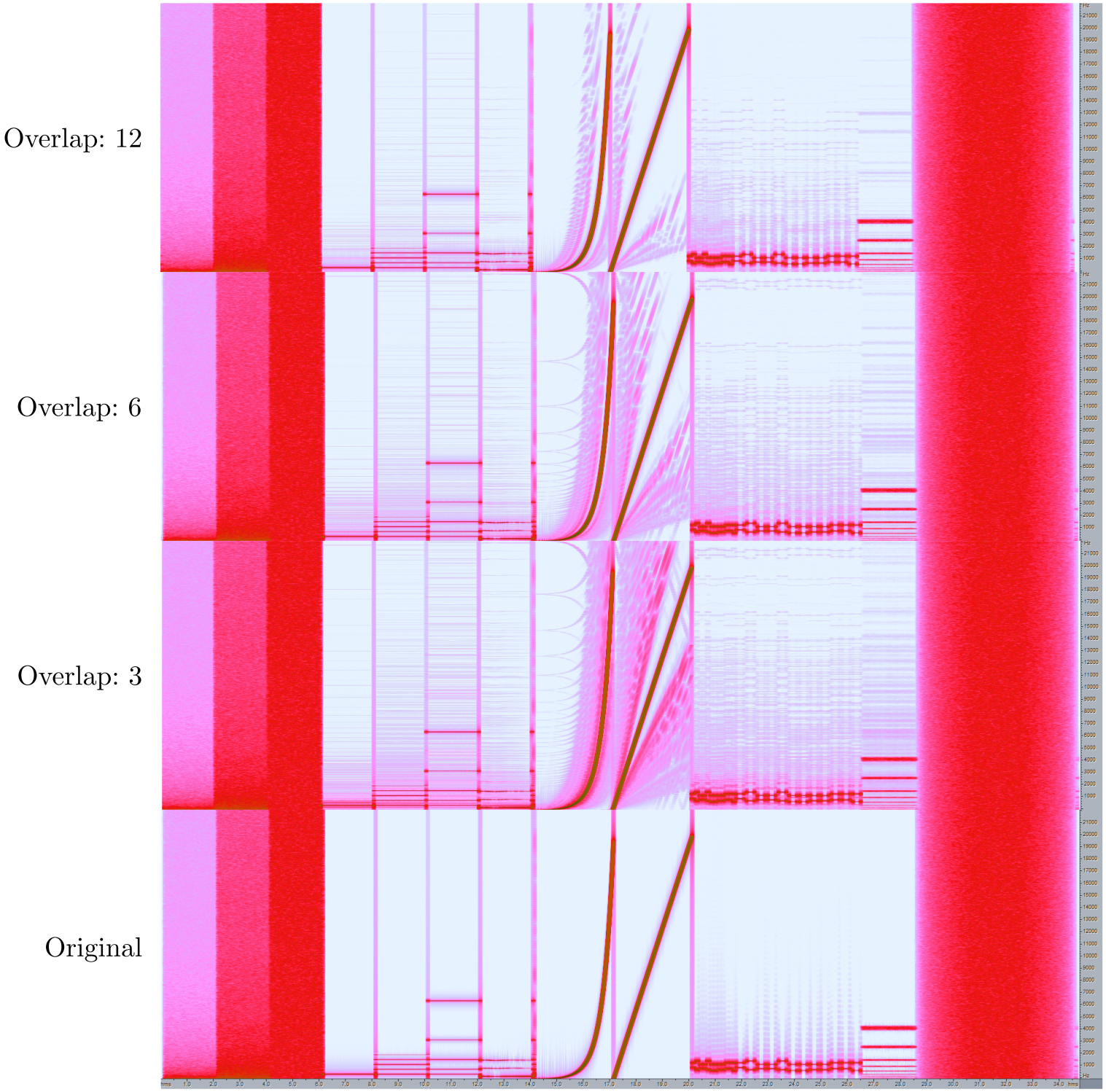
Overlap: 12

Overlap: 6

Overlap: 3

Original



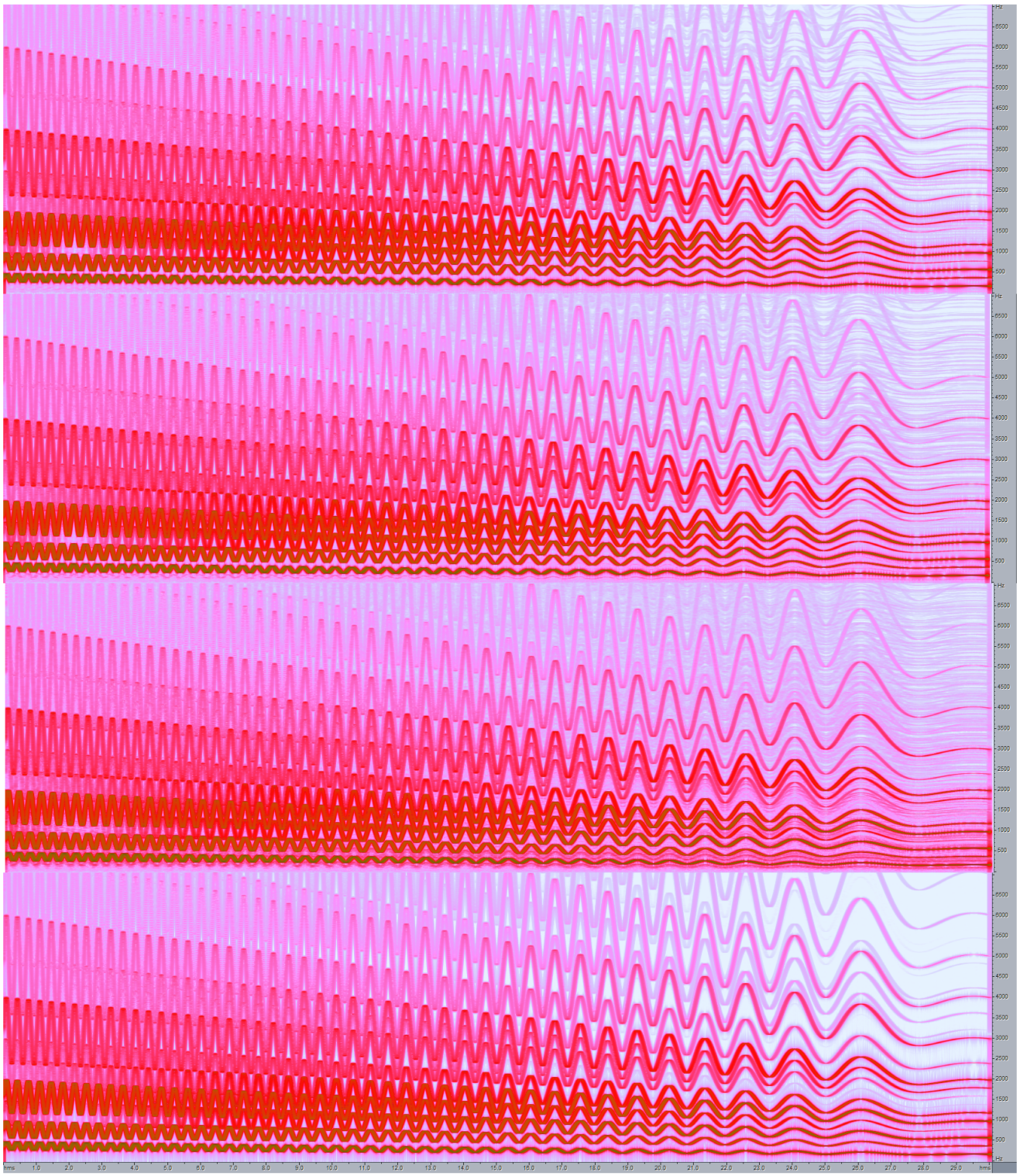

Overlap: 12
Overlap: 6
Overlap: 3
Original

Overlap: 12

Overlap: 6

Overlap: 3

Original

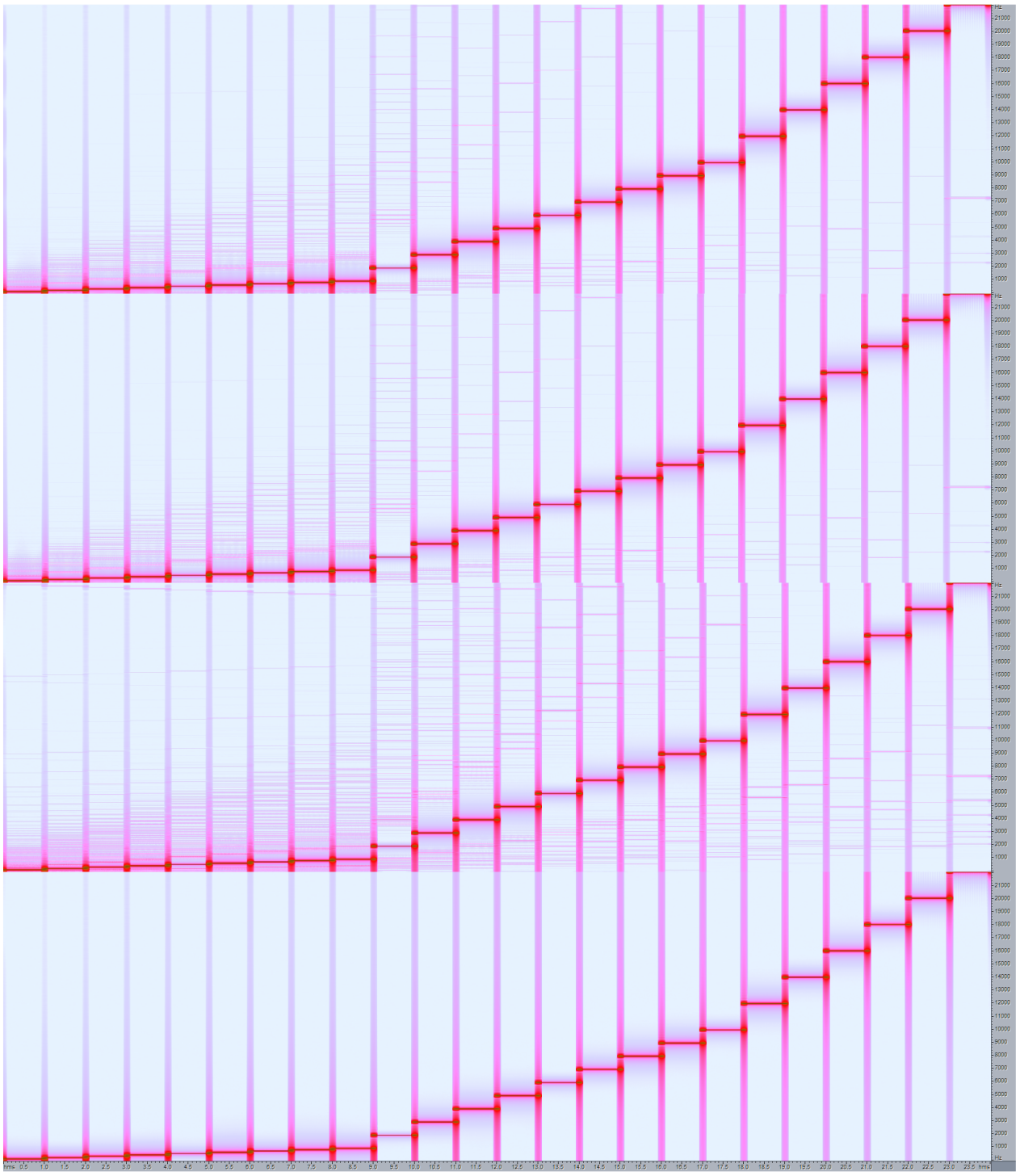


$\mathrm{WL}_c = 0.75s$
(factor 4)

$\mathrm{WL}_c = 0.375s$
(factor 2)

$\mathrm{WL}_c = 0.1875s$
(factor 1)

Original

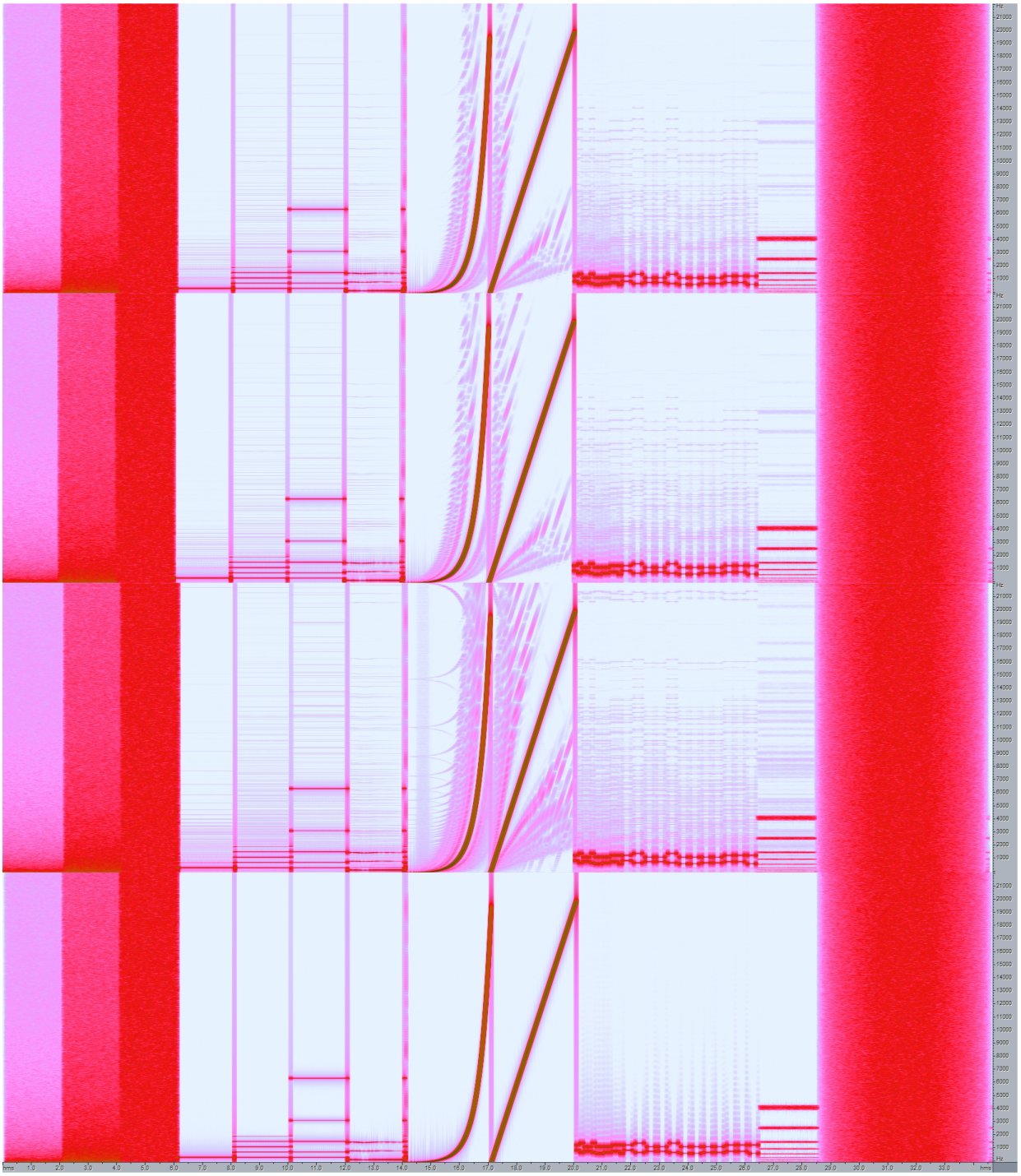


$\mathrm{WL}_c = 0.75s$
(factor 4)

$\mathrm{WL}_c = 0.375s$
(factor 2)

$\mathrm{WL}_c = 0.1875s$
(factor 1)

Original

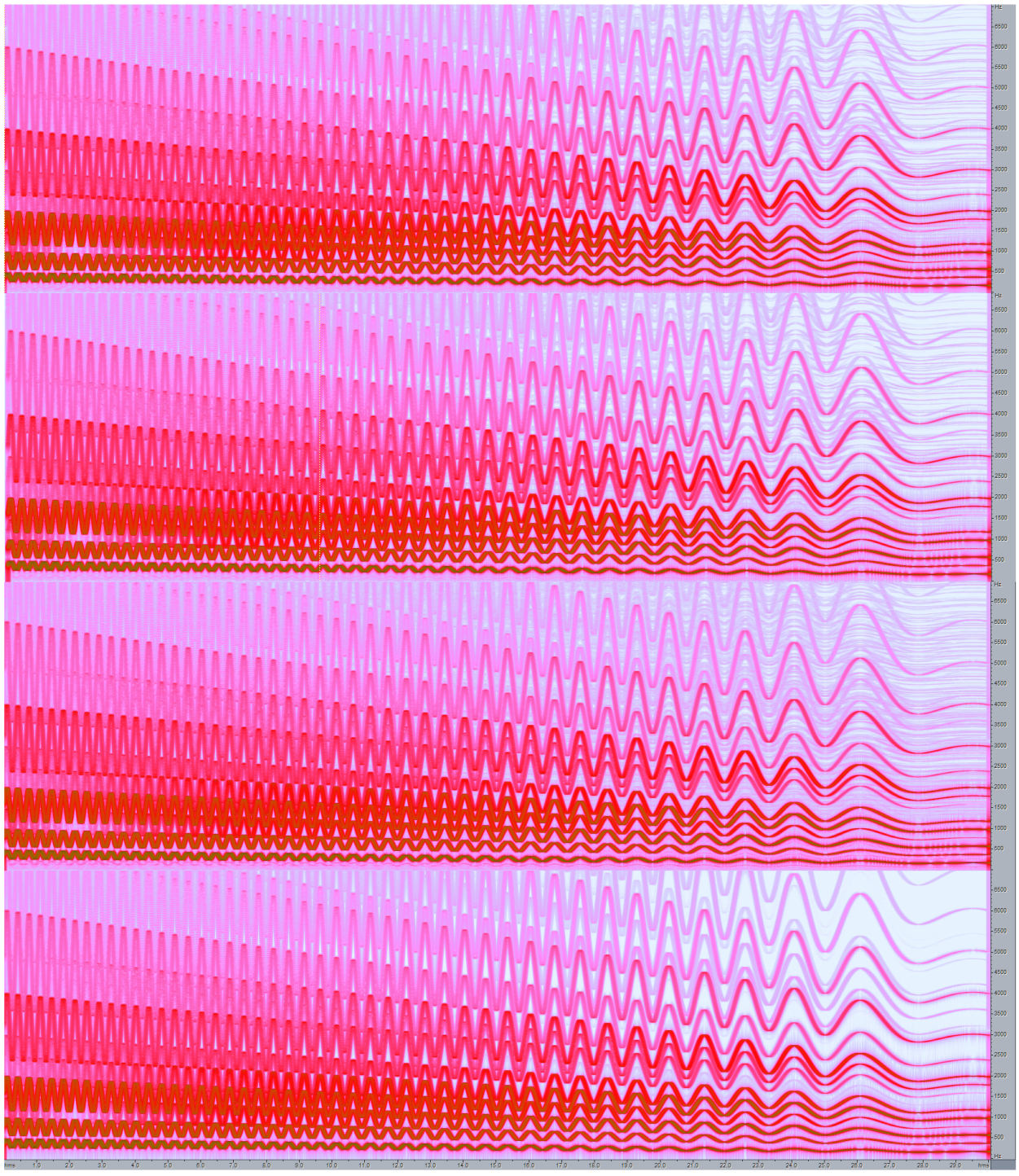


$\mathrm{WL}_c = 0.75s$
(factor 4)

$\mathrm{WL}_c = 0.375s$
(factor 2)

$\mathrm{WL}_c = 0.1875s$
(factor 1)

Original



$\sum_q \mathrm{WL}_q/N = 39684$

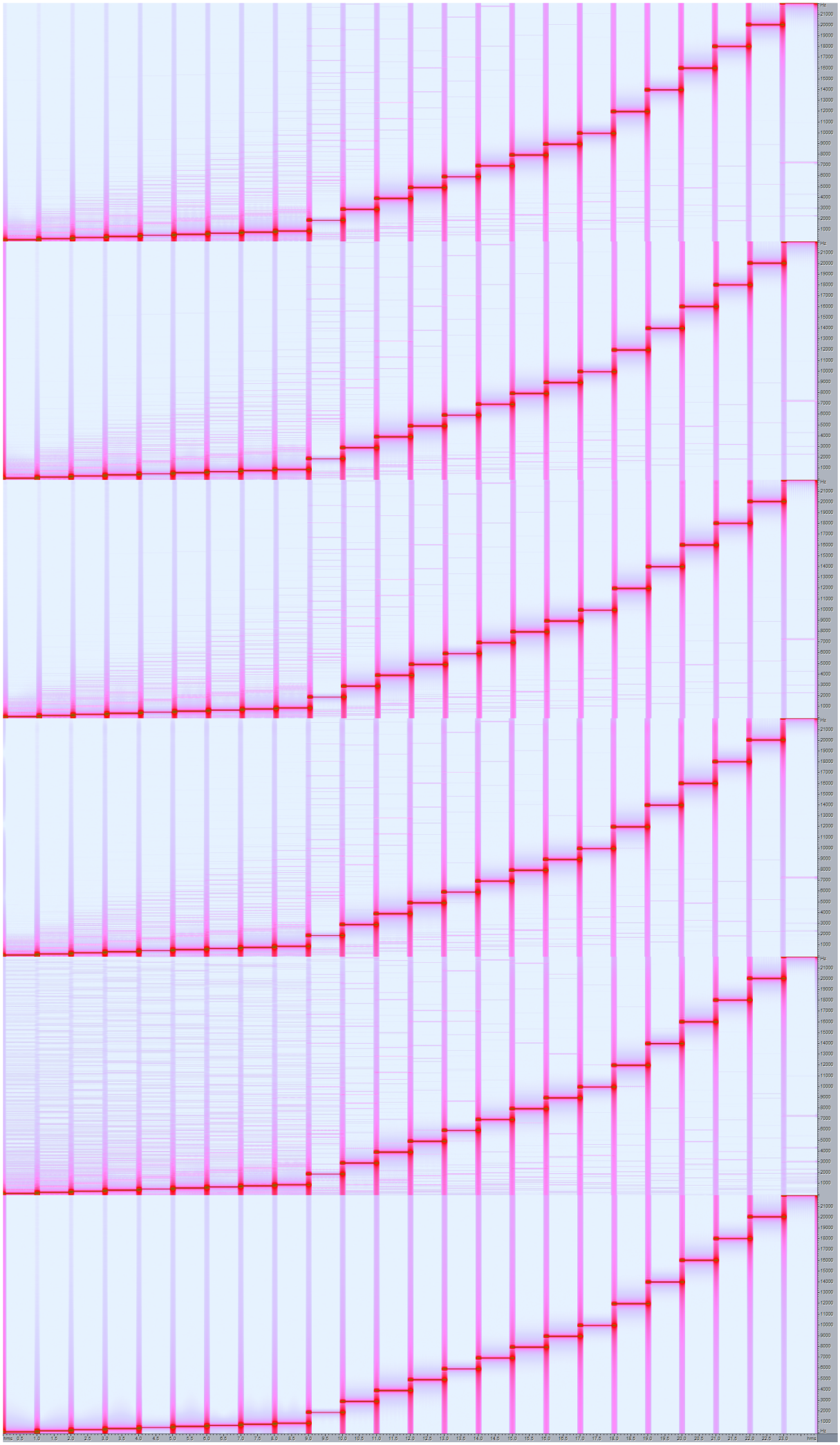
$\sum_q \mathrm{WL}_q/N = 14193$

$\sum_q \mathrm{WL}_q/N = 10753$

$\sum_q \mathrm{WL}_q/N = 7066$

$\sum_q \mathrm{WL}_q/N = 2577$

Original:



$\sum_q \mathrm{WL}_q/N = 39684$

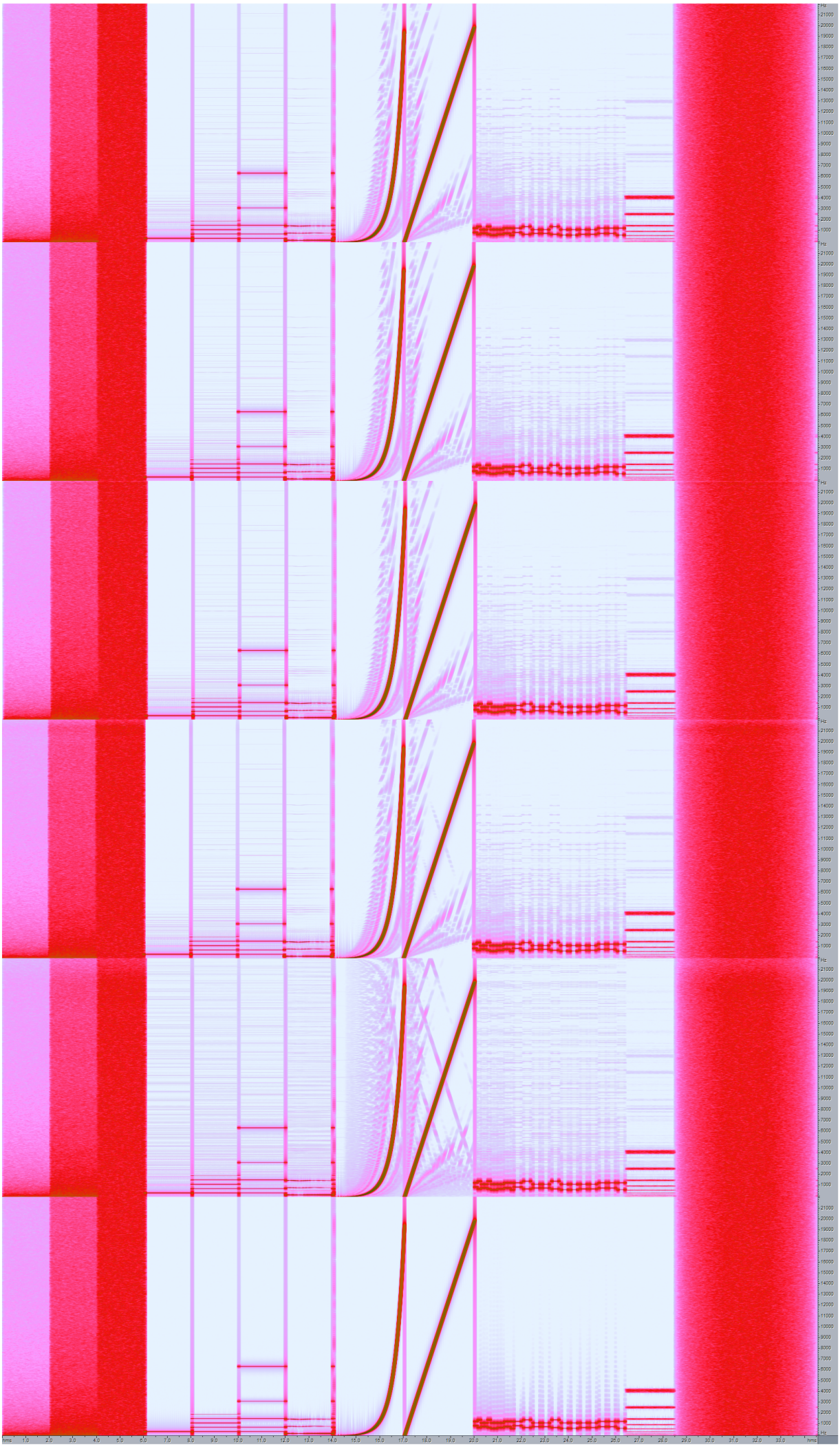
$\sum_q \mathrm{WL}_q/N = 14193$

$\sum_q \mathrm{WL}_q/N = 10753$

$\sum_q \mathrm{WL}_q/N = 7066$

$\sum_q \mathrm{WL}_q/N = 2577$

Original:



$\sum_q \mathrm{WL}_q/N = 39684$

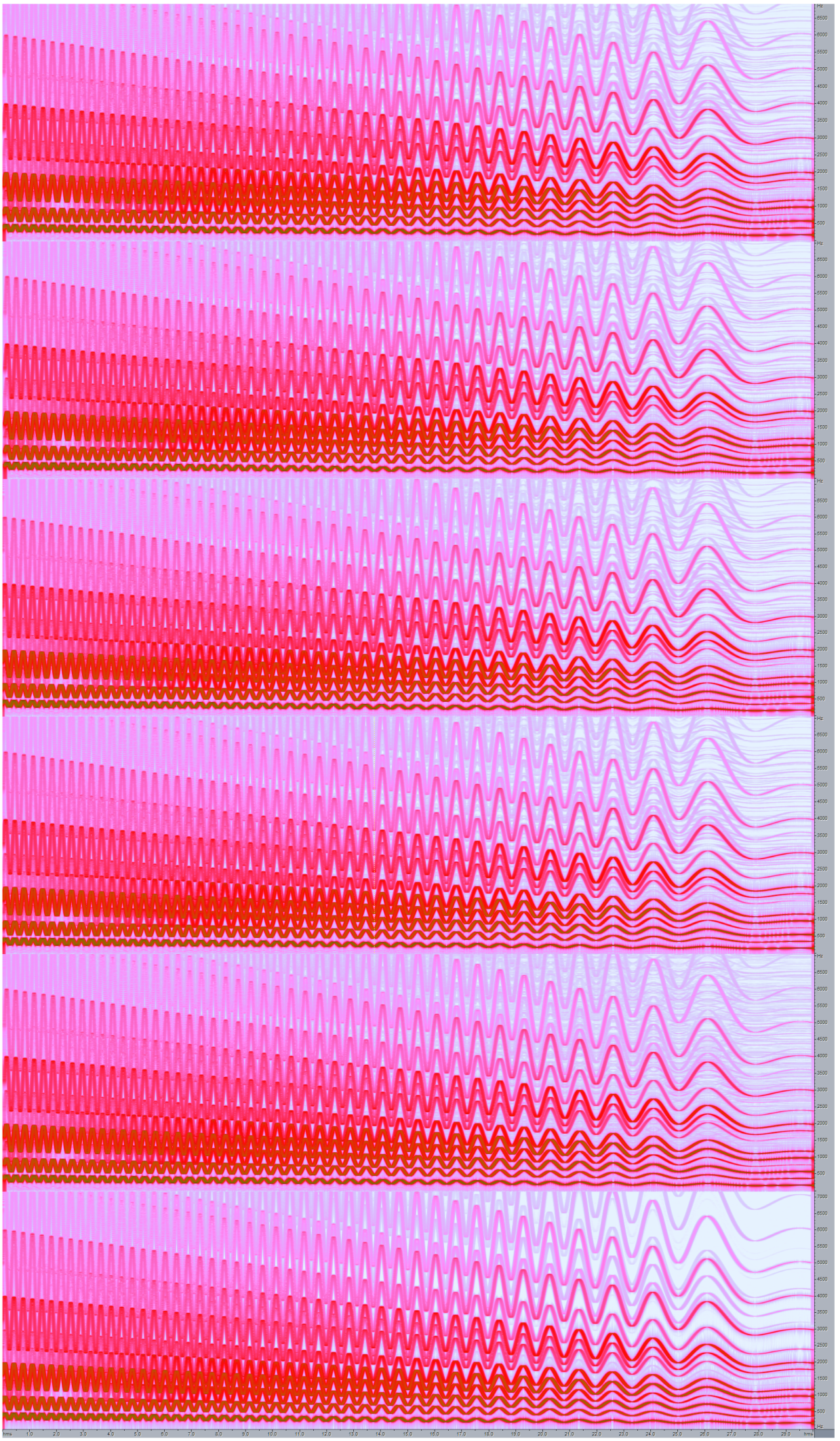
$\sum_q \mathrm{WL}_q/N = 14193$

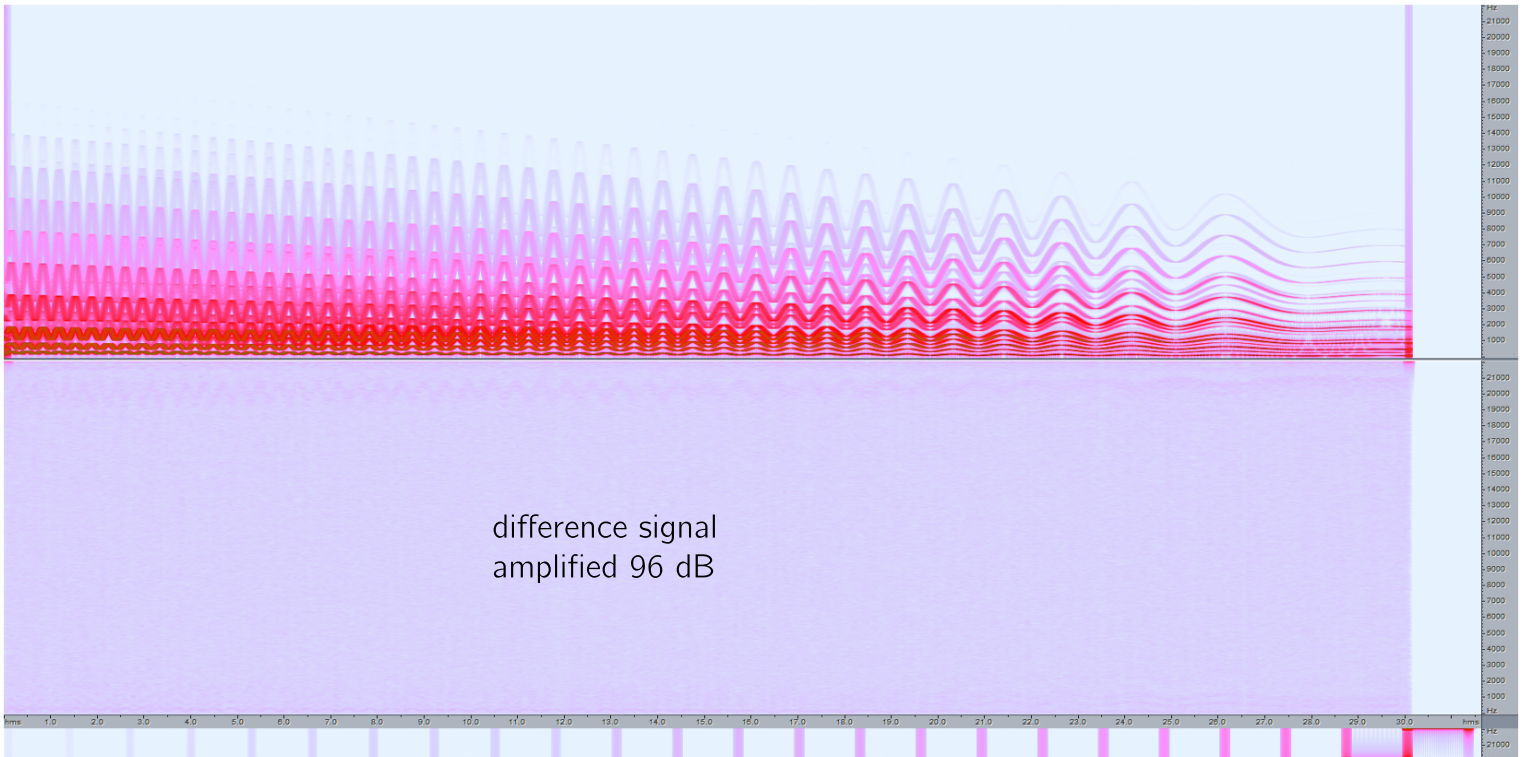
$\sum_q \mathrm{WL}_q/N = 10753$

$\sum_q \mathrm{WL}_q/N = 7066$

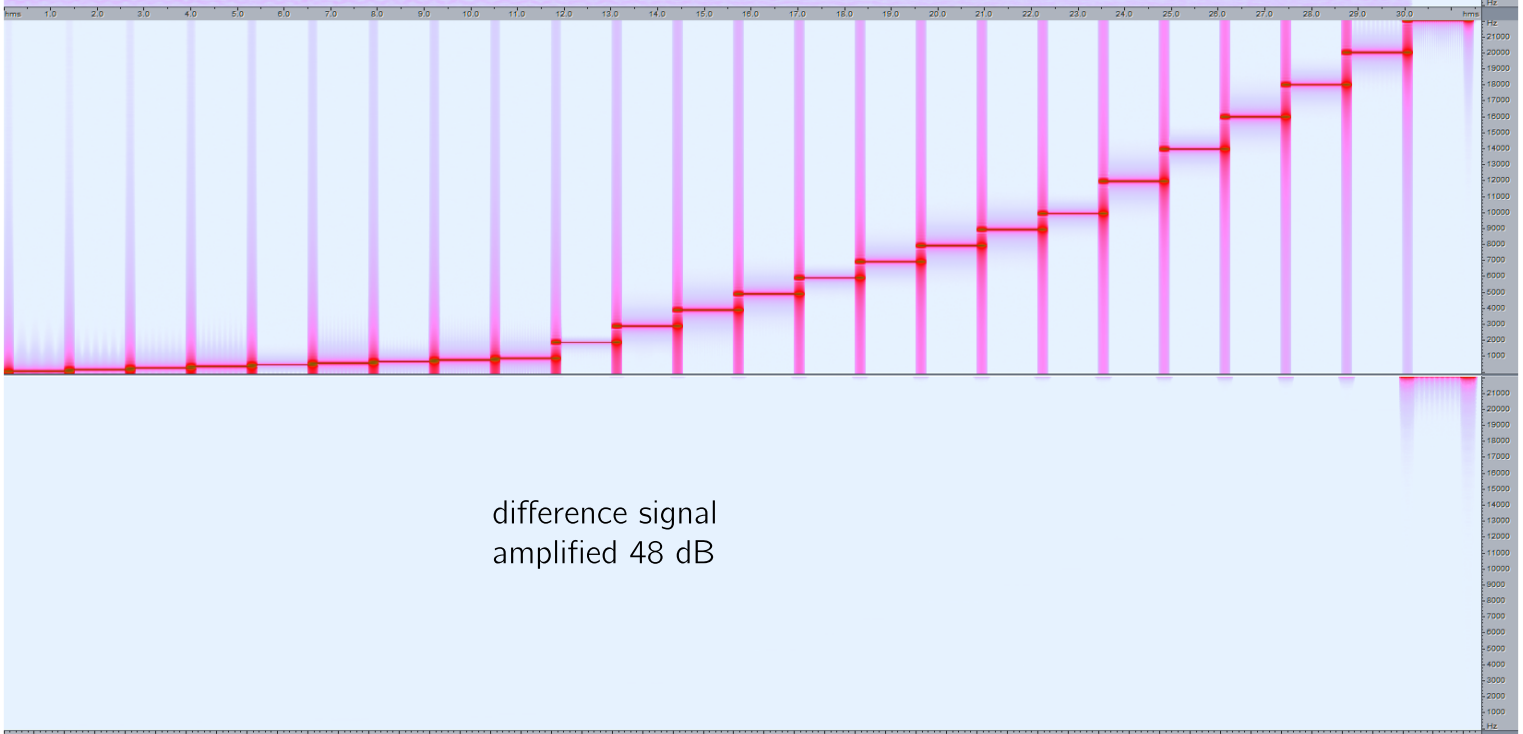
$\sum_q \mathrm{WL}_q/N = 2577$

Original:

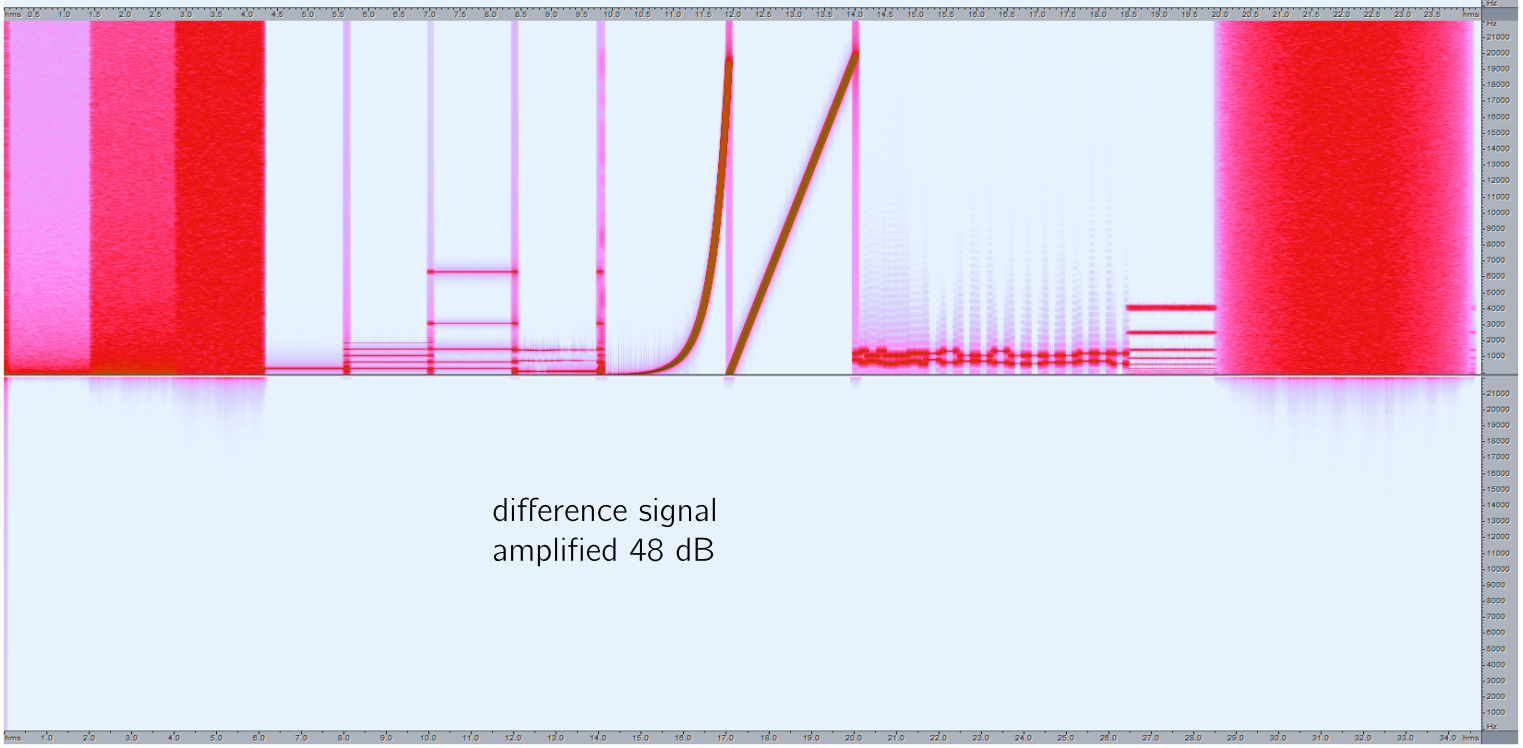




difference signal
amplified 96 dB



difference signal
amplified 48 dB



difference signal
amplified 48 dB

## 6.2 The pd-external

### Installation Guide for Windows

The file *wabor_t.dll* must be copied somewhere and the path to it must be passed to PD. For example, the file *wabor_t.dll* resides in `C:\plugins` than PD needs to be started with the command line argument `-lib C:\plugins\wabor_t`. The files *wabor~.pd*, *invwabor~.pd*, *welay~.pd* just need to be at a place where PD can find it. If PD says it cannot load the external, than probably the file *libfftw3l-3.dll* is missing in the folder `pd\bin\`.

### Source Code

I can send the source code on request by mail. My address is *tommsch@gmx.at*. The code is published under the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or any later version. This program is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose. See the GNU General Public License for more details.

### Interface

The file *wabor.dll* contains the following externals:

- *wabor_tilde*: The analysis part.
- *invwabor_tilde*: The synthesis part.
- *welay_tilde*: A quick and dirty delay line for arbitrary long delays. The delay is given in whole samples.

All objects come along with a wrapper patch, these are

- wabor~
- invwabor~
- welay~

which provide the control logic for the externals. Especially for the *wabor_tilde* external it is strongly recommended to use the wrapper. All objects do not receive any creation arguments. They are only for the purpose that the objects graphical box in PD is bigger. Furthermore the files

- *wabor.pd*: The patch which was used for the tests in this thesis
- *avg.pd:* A subpatch used in *wabor.pd* for computing averages.
- *wabout.pd*: A small “about” file.
- *wabor_min_example.pd*: The minimum example found below.
- *libfftw3l-3.dll*: The FFTW library for long double precision.
- *install.nfo*: A short installation guide.

should come along with *wabor.dll*.

Figure 6.1 shows the minimum example how to use the objects *wabor~* and *invwabor~*. *wabor~* has one sound input. The second inlet is for turning the computation of the coefficients on and off. The third and fourth inlet take floats and determine the band in which the signal gets analysed. This cut-off is quite exact. The rightmost inlet serves for sending messages direct to the wrapped external *wabor_tilde*. The leftmost outlet is for the Gabor Coefficients data, the right outlet for the control data. Both outlets must be connected to the respective inlets from *invwabor~*. The third inlet is for turning the computation of the coefficients on and off. The rightmost inlet of *invwabor~* is for messages direct to the wrapped *invwabor_tilde* external. The outlet of *invwabor~* carries the resynthesised sound. The rightmost outlet sends the delay in samples due to the analysis/synthesis part as a float. In the example this value is sent to *welay~* which delays the input from *adc~*. Afterwards the delayed original is subtracted from the analysed-synthesised signal and low-pass filtered with a roll of frequency of 20 kHz. Hence there should be no output.

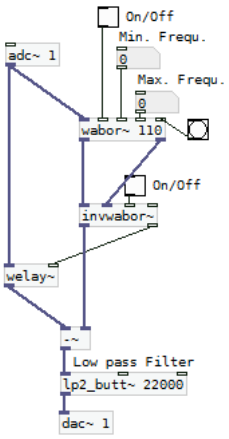



Figure 6.1: A minium patch showing the use of the *wabor_t* external.

### wabor~

Most of the subpatch is shown on page 76. It is split up in several parts.

**Section 1:** This is just the actual external. The output sends data to the array in section 9.

**Section 2:** This section is actually an abstraction inside the subpatch with the name *functions* and contains commands to manipulate the generated windows. The general format of the commands is *function-name array-number parameters ... [random]*. *Function-name* selects the function which shall be applied. All possible command are written below. *Array-number* depicts which array shall be manipulated, a value of $-1$ means that the command is applied to all windows. *Parameters* are one or more numbers, depending on the command. Some command take an

optional *random* parameter. If so, the outcome is random (like pointwise multiplied with a random number).

**cycle** shifts the whole array by an integer. Takes one additional integer or string and one optional int. The second parameter can be positive or negative, or of the following format "± N/x", where x is a number between 1 and 9. Then the array gets shifted by floor(N/x) bins, where $N$ is the windows length. Examples are: *cycle -1 N/2*, *cycle 10 -N/4*, *cycle -1 40*.

**mult** multiplies/divides/... the window pointwise with a number. Takes one additional string, two floats and one optional int. The second parameter defines what shall be done.

TIMES_VAL: multiplies everything with the complex number given via the following two floats. When appending a 1 as a third number, then everything is multiplied by a random number which has at most the magnitude of the given number.

SQUARE: Squares everything pointwise.

TIMES_N: multiplies pointwise by the window length (useful after applying a FFT).

DIV_N: Divides pointwise through the window length.

RECIPROCAL: Takes pointwise the reciprocal values.

CONJUGATE: Conjugates pointwise.

Examples are: *mult -1* TIMES_VAL *19 2.3 0*, *mult -1* DIV_N *0 0*

**cut** takes two additional floats. Everything which is in absolute value smaller than the first value gets scaled up to the length of the first value. Everything which is bigger than the second value gets scaled down to the length of the second value. For example: *cut -1 0 100*, *cut -1 1.2 1.3*

**fft** computes the forward FFT the window. The FFT does no normalization. Applying the FFT four times result in the window being scaled by $N^2$, where $N$ is the window length.

**set_nan_to_zero** sets every *NaN* in the windows to zero. Actually there should not be any NaN's.

**componentabs** maps every value $x = a + ib$ in the window to $x \mapsto |a| + i\,|b|$.

**ln** takes the pointwise logarithm

**abs** takes the pointwise absolute value

**normalize** normalizes the window, such that the maximum absolute value equals 1.

**differentiate** takes the difference quotient, $win(i) := win(i) - win(i-1)$.

**integrate** sums the window up, $win(i) := \sum_{j=0}^{i-1} win(j)$. After integrating the window is set to DC offset zero.

**imag_plus_real** maps every value $x = a + ib$ to $x \mapsto (a + b) + i(b - a)$.

**make_dc_offset** makes an DC offset. Takes four additional floats. If the second parameter is zero, then the current offset is only computed and printed in the PD main window. If it is one, then the window gets shifted so that it has a complex offset given by the last two floats.

**prolong** extends the window by inserting zeros. Takes two additional ints and one optional int. The second parameter is the number of zeros which shall be inserted. The third parameter determines the

place where the zeros shall be inserted. 0 means at the beginning, 1 in the middle, 2 at the end. If the last parameter is one, then the number of zeros which are inserted are random, but smaller than the passed value.

**Phase_in_Degrees:** This is no function, just a wrapper for the mult command. After banging, the window which is visible at the array (section 9), gets multiplicated with the complex number $e^{i\phi}$ where $\phi$ is given in degrees.

**Section 3:** **make** computes the windows. Takes three ints. The first determining the window which shall be computed, *-1* means all windows. If the third parameter is 1, then a memory saving procedure is used. In this case if the windows shall be shortened, the second parameter has to be one. If the last parameter is zero, then only the first parameter has an effect.

**shorten** shortens the window, can only be used if the non-memory-saving version was used to generate the windows.

**copy_pac2paf:** The reason for this behaviour is that I compute the windows with a high precision, storing them in the array `pac`. With the command *copy_pac2paf* the window gets copied to the array `paf` which is the array used for analysing/synthesising. The command *copy_paf2pac* copies the window data back from the `paf` array to the `pac` array.

With sections 4, 6 and 7 the parameters for window computing are set.

**Section 4:** With the bangs reasonable parameters can be set at once. **Set WIN_COS** sets an exponential warping map and a raised cosine window. **Set WIN_COS_not_warped** sets a standard STFT. Since the algorithm cannot use the FFT, analysis and synthesis for that will be very slow. **Set Start Values** sets the parameters for the start-up of the external, so that nothing goes wrong. The inside of this abstraction is of no big use afterwards, except for the included command **set_soundbuffer_length_factor** which sets the length of the sound buffer as a multiple of the PD block size (64 samples).

**Section 6:** Most of the commands are self explanatory. After setting some value the command *initialize* must be executed. This is done automatically for all commands in this section. After the use of *initialize* all windows are deleted and must be computed again.

**set_window_type** sets the used basic window. The raised cosine window has value 2 and is the only window available.

**set_theta_type** sets the used warping map.

EXP_CINF, EXP_C1, EXP_C0, EXP_DISCONT all are exponentially increasing functions. They differ in their behaviour around zero, since the warping map needs to be bijective and odd but $e^0 = 1$. CINF is $C^\infty$, C1 is $C^1$, C0 is continuous and DISCONT is discontinuous at zero. Exactly:

- $\theta_{discont}(f) = \begin{cases} 0 & \text{if} \quad f = 0 \\ \sigma(f) f_0 2^{\frac{|f|}{d}} & \text{otherwise} \end{cases}$

- $\theta_{C0}(f) = \begin{cases} \frac{2f_0}{d} f & \text{if} \quad |f| \leq 2d \\ \sigma(f) f_0 2^{\frac{|f|}{d}} & \text{otherwise} \end{cases}$
- $\theta_{C1}(f) = \begin{cases} \frac{f_0 \ln(2)}{d} f & \text{if} \quad |f| \leq d/\ln(2) \\ \sigma(f) \frac{f_0}{e} 2^{\frac{|f|}{d}} & \text{otherwise} \end{cases}$
- $\theta_{C\infty}(f) = \sigma f_0(f) 2^{\frac{f^2-1}{fd}}$

ID is the identity map $\theta_{id}(f) = f$.
LIN is the map $\theta_{lin}(f) = f_0 f$.
SQUARE is the map $\theta_2(f) = df^2$.
The variables $f_0$ and $d$ are parameters.

**set_compute_type** sets whether the windows should be computed in the linearised version (1) or not (0).

**Section 7:** The commands in this section should not be used and are only there for debugging. They set some parameters directly, but not in a straight forward way. The *window_type* must not be 2 if one wants to use these commands.

**Section 5:** This functions can be used without the need to delete and recompute the windows. Most of them are self explanatory.

**set_cut_ratio** sets a parameter of how much the windows should be cut. The higher the value, the less cutting.
**set_threaded** sets whether multi-threading is used or not. After changing this value Pure Datas DSP engine must be stopped and started again.
**set_no_wigwag** If this is set, the numerical errors in the windows near $t = 0$ are reduced by filtering the windows.
**set_winlength_max** sets the maximum length of the windows. This is useful if a desired maximum latency is required.

**Section 8:** This section contains commands to get information about the computed windows. This is probably the most useful part of the subpatch. Most commands expect one parameter denoting the window which is prompted, again: $-1$ means all windows.

**bang** prints all parameters, prints more if double clicked.
**array_synopsis** gives a summary over the windows.
**print_brand** gives a summary over the windows essential bandwidths; very interesting.
**print_first_and_last_value** prints the first and last value of the window
**inner_product** computes the inner product, takes two parameters which are the band numbers. $-1$ is possible in both arguments. *inner_product -1 -1* takes a long time to compute.
**length_of_vector** computes $\sqrt{\langle win(i), win(i) \rangle}$ and prints the result in the PD main window.
**send_window_to_array:** With this number box one can examine the computed windows, very comforting.

**Section 9:** The PD arrays *array_wabor_all* and *array_wabor_zoom* for contemplation with the windows. There is a PD bug resulting in frequently

crashing PD when closing this patch. If the two arrays are deleted, then the bug does not arise.

**invwabor~**

The *invwabor~* subpatch is self explanatory. There are only three commands for the *invwabor_tilde* external, namely

**set_soundbuffer_length_factor** sets the length of the used sound buffer as a factor of the block size.

**set_compute_synthesis** enables or disables the synthesising.

**set_threaded** sets whether multi-threading shall be used or not.

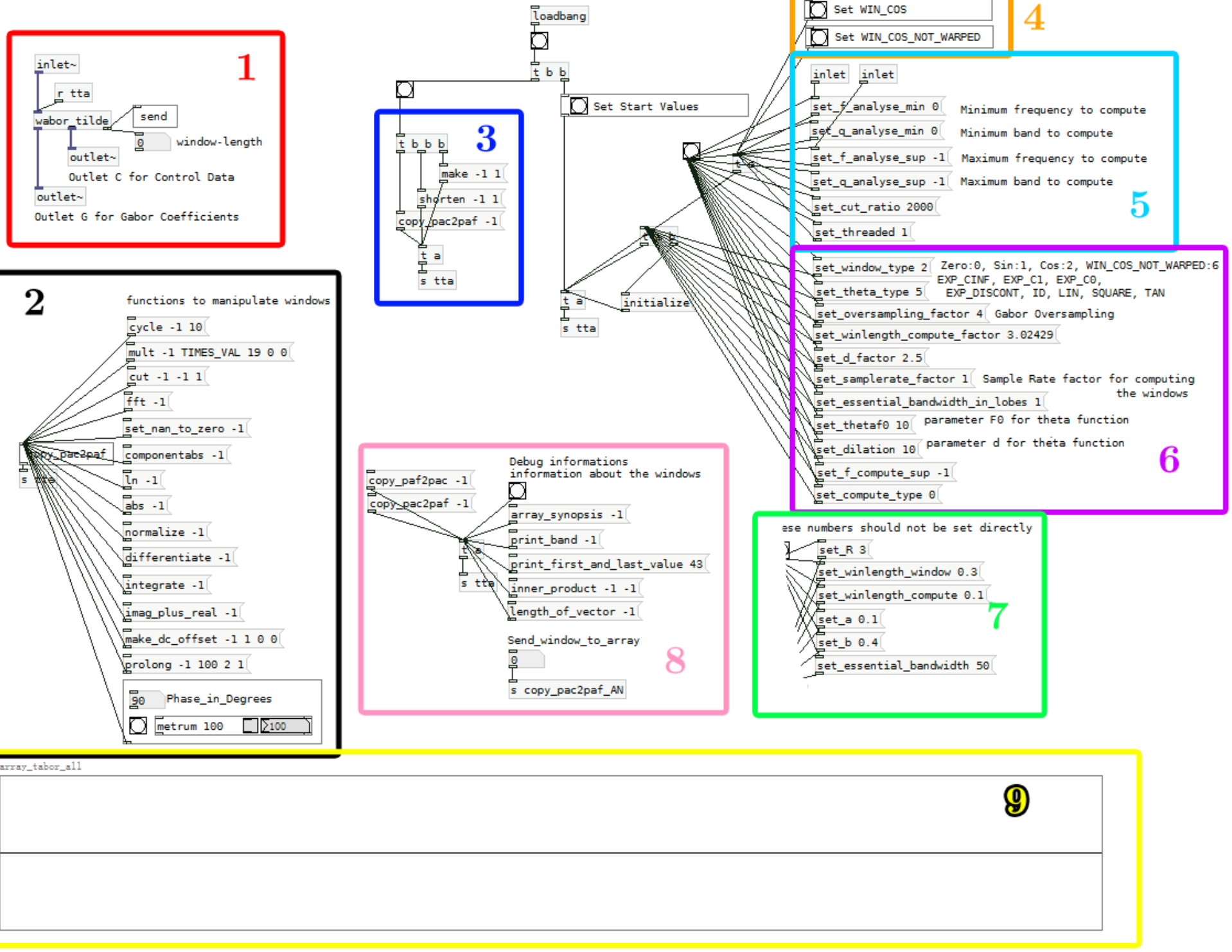
The second sound output is unused. The float inlet is also unused.

**welay~**

The same applies for the *welay~* subpatch.

## Already known bugs

- Closing a patch which contains an array object can crash Pure Data. (This is PD Bug.)
- Not all of my functions prove against NULL Pointers before accessing memory. Nevertheless, normal use of the external will not cause any problems.
- Sometimes a message must be sent two or three times to the wabor externals until they accept it.
- Since only one window function is implemented so far, the only admissible value for **set_theta_type** is 2.
- In the main window a lot of error messages may appear. Most of the times they are meaningless.
- If one uses multi-threading, the alignment of the windows to the signal is wrong if the sound buffer is full. This leads to a small displacement every time the buffer is full, thus renders measurements of the error meaningless.


1
inlet~
r tta
wabor_tilde
send
0
window-length
outlet~
Outlet C for Control Data
outlet~
Outlet G for Gabor Coefficients
2
functions to manipulate windows
cycle -1 10
mult -1 TIMES_VAL 19 0 0
cut -1 -1 1
fft -1
set_nan_to_zero -1
componentabs -1
ln -1
abs -1
normalize -1
differentiate -1
integrate -1
imag_plus_real -1
make_dc_offset -1 1 0 0
prolong -1 100 2 1
90
Phase_in_Degrees
metrum 100
100
3
loadbang
t b b
Set Start Values
t b b b
make -1 1
shorten -1 1
copy_pac2paf -1
t a
s tta
t a
initialize
s tta
4
Set WIN_COS
Set WIN_COS_NOT_WARPED
5
inlet
inlet
set_f_analyse_min 0
Minimum frequency to compute
set_q_analyse_min 0
Minimum band to compute
set_f_analyse_sup -1
Maximum frequency to compute
set_q_analyse_sup -1
Maximum band to compute
set_cut_ratio 2000
set_threaded 1
6
set_window_type 2
Zero:0, Sin:1, Cos:2, WIN_COS_NOT_WARPED:6
EXP_CINF, EXP_C1, EXP_C0,
set_theta_type 5
EXP_DISCONT, ID, LIN, SQUARE, TAN
set_oversampling_factor 4
Gabor Oversampling
set_winlength_compute_factor 3.02429
set_d_factor 2.5
set_samplerate_factor 1
Sample Rate factor for computing
the windows
set_essential_bandwidth_in_lobes 1
set_thetaf0 10
parameter F0 for theta function
set_dilation 10
parameter d for theta function
set_f_compute_sup -1
set_compute_type 0
7
ese numbers should not be set directly
set_R 3
set_winlength_window 0.3
set_winlength_compute 0.1
set_a 0.1
set_b 0.4
set_essential_bandwidth 50
8
copy_paf2pac -1
copy_pac2paf -1
t a
s tta
Debug informations
information about the windows
array_synopsis -1
print_band -1
print_first_and_last_value 43
inner_product -1 -1
length_of_vector -1
Send_window_to_array
0
s copy_pac2paf_AN
array_tabor_all
9

## 6.3 Pure Data

Pure Data is an open source visual programming language, mainly written and maintained by *Miller S. Puckette* and can be downloaded from:

- *https://puredata.info/downloads.*

Manuals for Pure Data can be found at:

- *https://puredata.info/docs*
- *https://www.flossmanuals.net/pure-data/*
- *http://pdstatic.iem.at/*
- *http://msp.ucsd.edu/Pd_documentation/*

A good tutorial written by *IOhannes m zmölnig* about how to write externals for PD can be found at:

- *http://pdstatic.iem.at/* or better, by a search for: *HOWTO External Pure Data.*

A guide written by *Leonardo/modlfo* about how to compile externals with Microsoft Visual Studio can be found at:

- *https://nontranscendentalexistence.wordpress.com/2012/07/27/developing-pure-data-extensions-in-visual-studio/*

A sample makefile for compiling externals with gcc is:

- ```
  CFLAGS := -mms-bitfields -DPD -DVERSION='0.0'
  CFLAGS += -g -g3 -Wall -W
  LDFLAGS := -shared -Wl,--enable-auto-import -g -Werror
  LDFLAGS += -L./pd/bin -L./fftw
  LDFLAGS += wabor_t.o libfftw3-3.dll pd.dll
  #---------------------------------------
  all : wabor_t.dll

  wabor_t.dll: wabor_t.o
          gcc $(LDFLAGS) -o wabor_t.dll

  wabor_t.o : wabor_t.c wabor_t.h
          gcc $(CFLAGS) -c wabor_t.c -o wabor_t.o
  ```
  ```

# Bibliography


[1] Richard G. Baraniuk and Douglas L. Jones. Unitary Equivalence and Signal Processing. In *International Symposium on the Mathematical Theory of Networks*, pages 617–620, Aug. 1993.

[2] John J. Benedetto, Christopher Heil, and David F. Walnut. Differentiation and the Balian-Low Theorem. *Journal of Fourier Analysis and Applications*, 1(4):355–402, 1994.

[3] Helmut Bölcskei. Gabor expansion and frame theory. Master's thesis, Vienna University of Technology, September 1994.

[4] P. W. Broome. Discrete orthonormal sequences. *Journal of the ACM*, 12:151–168, April 1965.

[5] Ronald Raphael Coifman, Yves Meyer, Stephen R. Quake, and Mladen Victor Wickerhauser. Signal processing and compression with wavelet packets. In James S. Byrnes, Jennifer L. Byrnes, Kathryn A. Hargreaves, and Karl Berry, editors, *Wavelets and Their Applications*, volume 442, pages 363–379. Kluwer Academic Publishers, Dordrecht/Boston/London, 1994.

[6] Ingrid Daubechies. *Ten Lectures on Wavelets*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.

[7] Gianpaolo Evangelista. Approximations for online computation of redressed frequency warped vocoders. In *Proceedings of the 17th International Conference on Digital Audio Effects, DAFx-14, Erlangen, Germany, September 1-5, 2014*, pages 85–91, 2014.

[8] Gianpaolo Evangelista. Lecture notes: Programmieren für Musiker, 2015.

[9] Gianpaolo Evangelista, Monika Dörfler, and Ewa Matusiak. Arbitrary phase vocoders by means of warping. *Musica/Tecnologia*, 7, 2013.

[10] D. Gabor. Theory of Communication. *J. IEE*, 93(26):429–457, November 1946.

[11] A.J.E.M. Janssen. Signal analytic proofs of two basic results on lattice expansions. *Applied and Computational Harmonic Analysis*, 1(4):350–354, 1994.

[12] L. Knockaert. On Orthonormal Muntz-Laguerre Filters. *IEEE Transactoions on Signal Processing*, 49(4):790–793, April 2001.

[13] Henrique S. Malvar. Lapped transforms for efficient transform/subband coding. *IEEE Trans. Acoustics, Speech, and Signal Processing*, 38(6):969–978, 1990.

[14] P.B. Osgood. *Lecture Notes for EE 261 the Fourier Transform and Its Applications*. CreateSpace Independent Publishing Platform, 2014.

[15] Eva Wesfreid and Mladen Victor Wickerhauser. Signal processing via fast Malvar wavelet transform algorithm. *14em Colloque Groupe d'Etudes du Traitement du Signal et des Images*, pages 377–380, 1993.

[16] J. Wexler and S. Raz. Discrete Gabor Expansions. *Signal Process.*, 21(3):207–220, October 1990.

[17] Mladen Victor Wickerhauser. Acoustic signal compression with wavelet packets. In Charles K. Chui, editor, *Wavelets: A Tutorial in Theory and Applications*, pages 679–700. Academic Press Professional, Inc., San Diego, CA, USA, 1992.

# Curriculum Vitae

**Personal Data:**


Mag. Thomas Mejstrik BA

Born: $10^{th}$ December 1984 in Vienna

Nationaliy: Austria

Telephone Number: 0043-677-61144882

Email: thomas.mejstrik@gmx.at


**Education:**

March 2013 – June 2016 (presumably): Studies of „Music education - Voice and Instruments - Piano“ with Prof. M. L. Winter. Topic of Master Thesis: *Real Time Computation of Redressed Frequency Warped Gabor Expansion*, Prof. Dr. G. Evangelista

October 2008 – March 2013: Bachelor in „Music education - Voice and Instruments - Piano“, University of Music Vienna with Prof. J. Marian, specialization in accompaniment and composition, graduated with distinction

since October 2008: Bachelor studies of Sinology, University of Vienna

October 2004 - December 2010: Master in Mathematics, University of Vienna, graduated with distinction. Title of Master Thesis: *Some Remarks on Nagumo's Theorem*, Prof. Dr. A. Constantin

June 2004: Secondary school, school leaving exam with distinction at the HTBLuVA Mödling, subject of main interest: precision engineering

1989 – 1999, 2002 – 2007: Studies of the piano at the music school in Korneuburg (Austria) with Lovorka Schenk.

**Studies Abroad**

October 2013 – June 2014: Masterstudies Piano Performance at the Central Conservatory for Music, Beijing （中央音乐学院，北京，中国） with Youxi （由熹）

October 2007 – June 2008: Chinese Studies, Central China Normal University in Wuhan, China （华中师范大学，武汉，中国）

**Publications:**

Thomas Mejstrik, *Some Remarks on Nagumo's Theorem*, Czechoslovak Mathematical Journal Volume 62, Number 1, 235-242

**Work Experience:**

since October 2012: Consultant at the bureau for women policy of the Austrian National Student Union at the University of Music Vienna

October 2013 - June 2014: Piano teacher at „Für Elise"-Center Beijing

March 2009: Exhibition *Imaginary* at the University of Vienna, general assistance

March - June 2008: Bar-pianist in Wuhan

September 2004 - December 2006: SFS-intec, abourer

July 2001: TECWINGS, Assistant at the development department

August 2002: SMC Pneumatik GesmbH, Assistant at the development department

**Skills:**

Windows, Latex, C++, C, VB, Linux (basic knowledge) and others

Good understanding of engineering and technology

Ballroom Dancing, Ballet

Composition

Driving License B

**Languages:**

German: mother-tongue

English: fluent

Chinese: moderate

French: basic

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit bis auf die offizielle Betreuung selbst und ohne fremde Hilfe angefertigt habe und die benutzten Quellen und Hilfsmittel vollständig angegeben sind.

Wien, den 10. November 2016