UNIVERSITY OF CALIFORNIA
Santa Barbara

Transfer: An Interactive Program for Real-Time Spectral Transformations and Visualization

A thesis submitted in partial satisfaction of the requirements for the degree
Master of Arts in Media Arts and Technology

by

Lance Jonathan Putnam

November 2005

ABSTRACT

Transfer: An Interactive Program for Real-Time Spectral Transformations and Visualization

by

Lance Jonathan Putnam

Sound transformation in the spectral domain is a powerful tool for composers and performers of computer music due to its inherent feature of separating the time and frequency information of sound. Spectral domain operations can now be executed in real-time with interactive control on modern computers. However, the lack of a well-rounded, intuitive interface for controlling these processes on multiple levels of interaction is a known problem. Transfer is a real-time spectral transformation and visualization program with a graphical interface that allows the creation of a modular signal processing graph for modifying the spectral data. The principle behind Transfer is that common tasks should be simple and intuitive to accomplish, so that more time can be spent doing creative work. Transfer offers processing modules in six distinct categories: magnitude, frequency, phase, peak, time, and utility. As an example, time-stretching and pitch-shifting can be accomplished by using time and frequency 'scale' modules. Other typical spectral processing algorithms such as low-/high-/band-pass filtering, noise reduction, and convolution are also easily accomplished. The ability to arrange the flow of spectral data processing and visualize the data in real-time provides instantaneous aural and visual feedback enabling rapid experimentation into new ways of modifying sound.

Contents

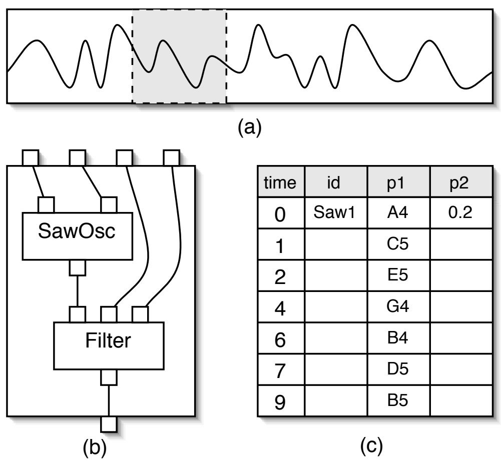
Chapter 1

Introduction

Sound transformation in the spectral domain is a powerful tool for composers and performers of computer music due to its inherent feature of separating the time and frequency information of sound. The phase vocoder is a popular technique in the computer music field for doing spectral-based sound transformation. Spectral domain operations can now be executed in real-time with interactive control on modern computers, however, no well-rounded, intuitive interfaces for controlling these processes on multiple levels of interaction are to be found. Much of the existing software interfaces for working in the frequency domain do not allow musicians, both composers and performers, to fully utilize its sonic potential.

As computers increase in speed and memory, an increasing number of algorithms and ways of processing data have become capable in real-time. Real-time operation is important for quickly assessing the behavior of algorithms, exploring large amounts of data, and having interactive control of systems. Unfortunately, doing these things in an intuitive manner without sacrificing flexibility of interaction is a difficult problem. The field of computer music faces these issues as do many other fields.

Human operators of computer software tend to have difficulty working with processes on multiple levels of details simultaneously. In the context of computer music, these levels of interaction lie on a continuum of timed events but can be segmented into nine distinct scales demarcated by perceptual boundaries [15]. For simplicity, only three levels of interaction will be used to evaluate software interfaces- sample, module, and sequence. Interaction on the sample level involves the manipulation of raw sound data described by pressure in the time domain and/or frequency, phase and magnitude in the frequency domain. The next level of interaction is marked by specification of signal flow between sound processing modules with input and output ports. This follows the concept of the unit generator conceived by Max Matthews at Bell Labs. The final level of interaction, the sequence, is concerned with the timed triggering of musical events. Sequences are used to execute things such as instantiation of concurrently running modules, modification of module parameters, and rearrangement of module connections. The sequence has a resemblence to what is historically known as the musical score.



time	id	p1	p2
0	Saw1	A4	0.2
1		C5	
2		E5	
4		G4	
6		B4	
7		D5	
9		B5	

Figure 1.1: Interaction at the level of the (a) sample, (b) module and (c) sequence.

Chapter 2

Background

2.1 The Phase Vocoder

The vocoder (voice coder), developed by Homer Dudley in the 1930s, was the first electronic device to encode and decode human speech. His vocoder encoded a voice signal by sending it through a parallel bank of band-pass filters. The output of the filter bank determined the energy of the signal at each filter's center frequency. This type of vocoder that uses a bank of band-pass filters is commonly referred to as a channel vocoder. The phase vocoder was originally proposed as an improvement over the channel vocoder for higher fidelity speech transmission [4], however, it has been largely ignored by the telecommunications field in favor of lower bandwith/quality techniques such as linear predictive coding. Computer musicians, in contrast, are more concerned with quality over bandwidth and have therefore welcomed the phase vocoder with open arms. The phase vocoder remains to this day a popular tool for sound analysis largely due to the computational efficiency of the underlying FFT and the intuitive nature of its basis functions, sinusoids. The term "phase" in phase vocoder comes from the fact that in addition to deriving the magnitudes of component sinusoids from a signal, as with a channel vocoder, you also get their phase. This extra information leads to better overall quality of resynthesis and time-scaling, as well as a convenient means for doing convolution via multiplication in the frequency domain.

The phase vocoder is a multi-step analysis-modification-resynthesis process based on the short-time Fourier transform (STFT), a DFT of overlapping windows of time domain samples. The STFT is used to improve the time resolution of the analysis, lessen the amount of spectral "smearing', and circumvent phase discontinuities during resynthesis. However, these improvements have a cost of generating more frequency domain data due to the redundancy of overlapping windows. This is commonly referred to as "data explosion" and naturally presents difficulties for computational efficiency and data storage. The following is a list of steps involved with the process of doing spectral transformations with the phase vocoder. Many classes of transformations require the spectral data be converted to another form, such as the magnitude/frequency form for

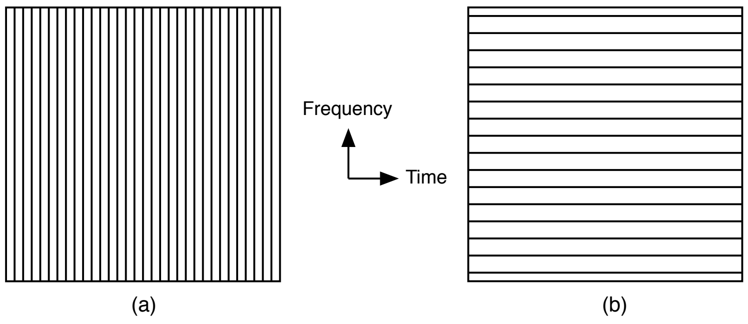


Figure 2.1: Time-frequency resolution grids of (a) time domain sampling and (b) frequency domain sampling (DFT). Each time-frequency energy band, signified by boxes, has the same area due to the the time and frequency resolution trade-off.

doing time-scaling.

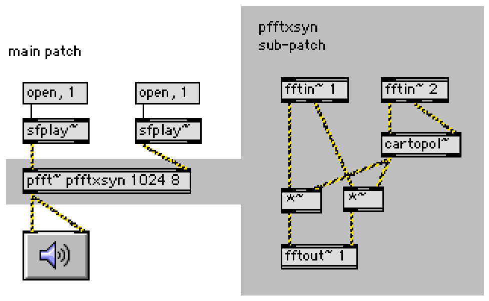
1. Analysis (STFT)
   a. Slide analysis window
   b. Perform FFT
2. Spectrum Modifications
   a. Convolution
   b. Data conversion (rectangular → polar)
   c. Magnitude/Phase transformations
   d. Data conversion (polar → magnitude/frequency)
   e. Frequency/Time transformations
3. Resynthesis
   a. Data conversion (?? → rectangular)
   b. Perform IFFT
   c. Window output time samples
   d. Overlap-add

The most interesting aspect of the phase vocoder, from a musician's point of view, is the stage between analysis and resynthesis where new sonic textures can be formed, transformation. The phase vocoder is a powerful tool for musical endeavor since it decouples the time and frequency information of a signal. This feature is most commonly exploited for pitch-scaling a signal without effecting its duration and changing the duration of a signal without modifying its pitch. Another major advantage of using the phase vocoder for manipulating sound is that it is quite straightforward to avoid frequency-aliasing, a caveat of many time domain transformations. This lends well to a less restrictive range of frequency-oriented transformations.

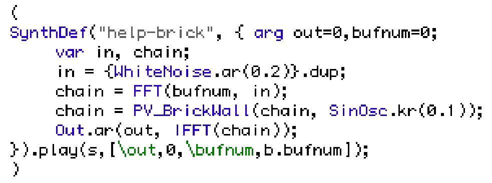
2.2 Existing Phase Vocoder Interfaces

2.2.1 Program Modules

Music programming environments such as CSound, Max/MSP, and SuperCollider supply a phase vocoder as multiple independent modules that do analysis, spectral modifications, and resynthesis. The user is expected to design a larger modular system that consists of one or more of these components. This model provides the musician with superior creative flexibility for musical processing since the interconnections between modules can be specified. The downside of this approach is that additional time must be spent as a systems and interface designer since modules are only purposeful within a larger context.



(a)



```
(
SynthDef("help-brick", { arg out=0,bufnum=0;
     var in, chain;
     in = {WhiteNoise.ar(0.2)}.dup;
     chain = FFT(bufnum, in);
     chain = PV_BrickWall(chain, SinOsc.kr(0.1));
     Out.ar(out, IFFT(chain));
}).play(s,[\out,0,\bufnum,b.bufnum]);
)
```

(b)

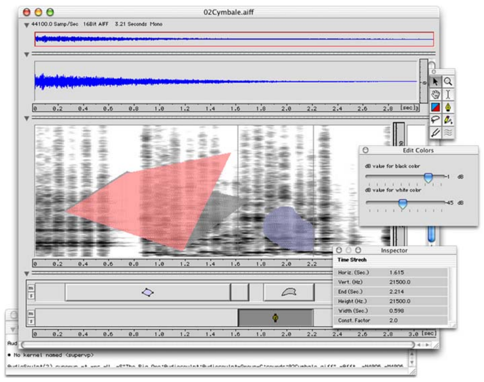
Figure 2.2: Phase vocoder modules in (a) Max/MSP and (b) SuperCollider 3.

2.2.2 Editors

There is clearly a need by many composers to have access to the sample level details of spectral data for doing sound transformations. The most successful interfaces for this are editor based ones that rely on a task-based mode of interaction through the keyboard and mouse. This form of interaction is based on the model/view/controller design pattern. Interactive control is generally focused on the sample level making this type of software unsuitable for a musical performance. One of the first software tools for doing spectral transformations was the Composers' Desktop Project (CDP) developed mainly by British composer Trevor Wishart. [17] Wishart has more recently created a graphical editor, Sound Loom, for using the sound transformations in CDP.

2.2.3 Plugins

A plugin is a module that is dynamically loaded at run-time by a host application to perform a specific processing task. Since the task of a plugin is well-defined and usually quite rigid, they can also offer an intuitive, graphical interface. This is what distinguishes a plugin from a module. Plugins have the advantage that they pre-package a commonly used configuration of lower-level modules and can be used 'out-of-the-box.' The disadvantage is that you


(a)

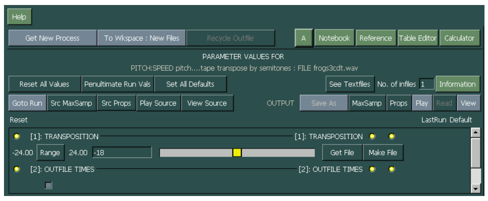
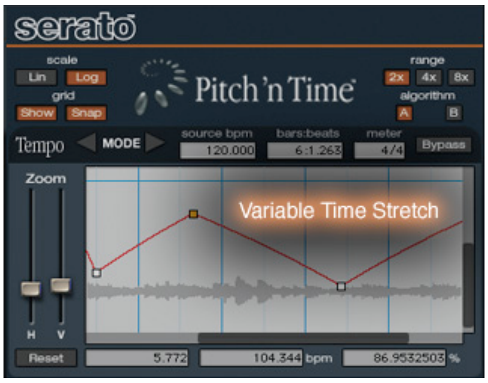
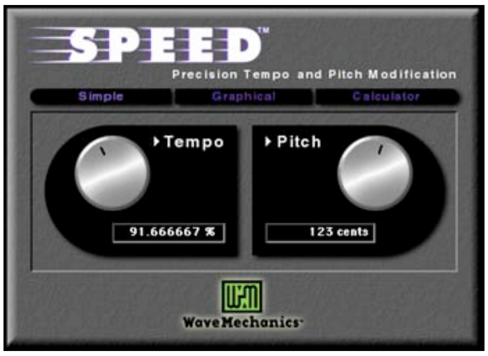

(b)

Figure 2.3: Screenshots from sample-level phase vocoder editors (a) Ircam's AudioSculpt and (b) CDP's Sound Loom.

depend on what the maker of the plugin thinks is an intuitive interface and many times only a limited range/detail of parameter values is possible. The most common types of spectral domain plugins are pitch- and time-scalers, noise-reducers, high-resolution graphic equalizers, and multi-band dynamics processors.



(a)



(b)

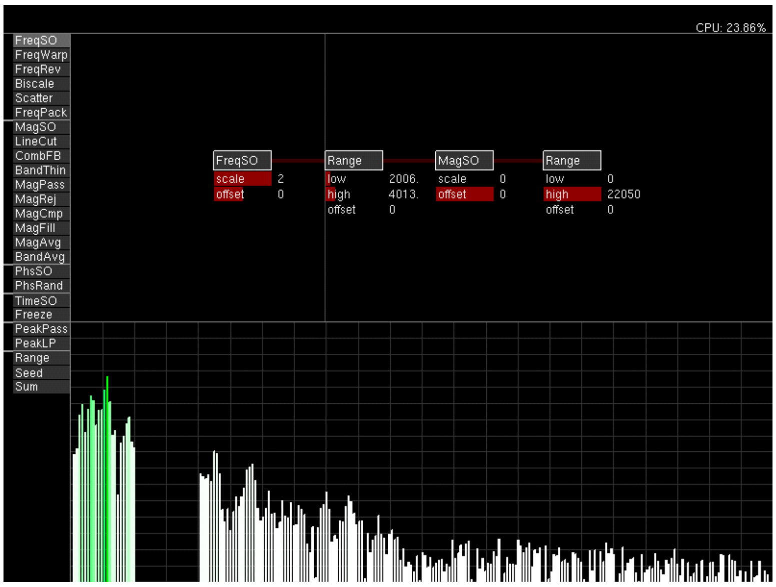
Figure 2.4: Examples of plugins that use spectral processing techniques. Serato's Pitch 'n Time (a) and SoundToy's Speed (b) are designed with a set purpose in mind and can therefore offer a more intuitive interface.

Chapter 3

Transfer Program Design

3.1 Overview

Transfer has a graphical, modular software interface for performing spectral transformations in real-time. The goal behind the development of Transfer was to design an interface that would enable a more complete and human-centered exploration of frequency domain transformations than existing software does. This research was focused on working on the module-level and visual modes of interaction.



```
FreqSO          Range            MagSO          Range
scale   2       low     2006.    scale   0      low     0
offset  0       high    4013.    offset  0      high    22050
                offset  0                       offset  0
```

Figure 3.1: Screenshot of Transfer.

The Transfer workspace is divided into three separate areas: the palette, the graph, and the visualizer. The palette, on the left side of the screen, is a list of the available spectral processing modules that can be instantiated into the signal processing flow. The graph region on the top right is where the user can modify the module parameters and arrange their execution order in the graph. The bottom right section is a scope for visualizing time domain and frequency domain representations of the processed signal.

3.2 Module Design

The basic underlying principle of Transfer is that spectral data is passed through a signal graph consisting of interconnected filtering modules. The modules are black boxes that take in an input signal, modify it according to an algorithm with n parameters, and output the modified signal. A diagram showing the behavior of a module is shown in Figure 3.2.

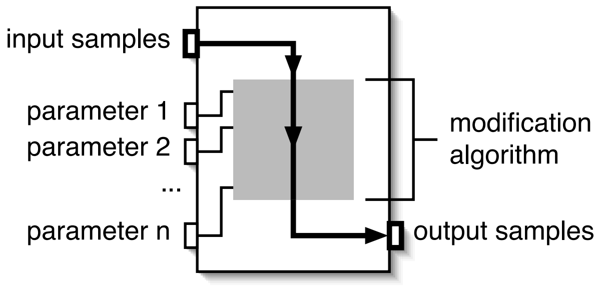
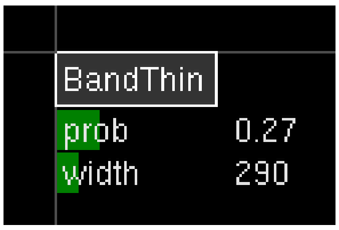


Figure 3.2: A signal processing module.

The module abstraction is important for allowing composers and performers to operate on sound in a higher-level, more intuitive manner through a small set of control parameters. Working with sound on the module level allows one to quickly perceive the effects of modifications and thus more easily shape the timbre of sounds.



BandThin
prob 0.27
width 290

Figure 3.3: Screenshot of a module icon within Transfer.

3.3 Catalog of Spectral Transformations

Spectral modules were divided into six unique categories in Transfer: magnitude, frequency, phase, time, peak and utility. The category of module indicates what effect the module has on the spectrum while its name was chosen to most accurately and concisely describe what the algorithm does. Magnitude modules are based on the simple multiplication of individual bins or groups of bin magnitudes by a certain transfer function. This class of transformation encompasses low-/high-/band-pass, spectral filtering, spectrum equalizing, noise-removal and -preservation and other forms of circular convolution. Frequency modules modify the frequency content of the spectrum by moving bin magnitudes around. This category includes processes such as pitch-shifting, frequency-shifting, and spectrum reversing. Phase modules apply operations to the phase information of the spectral data. This class of transformation includes chorusing, comb-filtering, smearing transients, and making robotic-like sounds. The time modules generally effect when spectral frames are played back. This category includes time-stretching and time-freezing. Peak modules operate on a higher level feature of the spectrum, the peaks, which are defined as bins whose magnitude is higher than its two neighboring bins. This class of effect is very similar to magnitude transformations, but tends to separate the sinusoidal components from the noise components of the sound. Finally, the utility class of module has special functions that control the behavior of downstream modules, such as selecting a certain frequency range to operate on or choosing a seed for random-based effects. Figure 3.4 shows a heirarchy of the different sound transformation modules, as well as their relation to existing techniques.

The next sections present a more detailed description of the modules implemented in Transfer.

3.3.1 Magnitude Transforms

This is a class of transformations that modifies only the magnitudes of the frequency samples. These are the "safest" type of transformations in that they will not affect the phase information of the signal and thus preserve the original transients. Many spectral domain processes tend to fall in this category of transformation including noise reduction, graphic equalization, cross-synthesis, and multi-band dynamics.

Band Thin

Randomly thin bands of frequencies.

Parameters:

prob- probability from 0 to 1 of a band being zeroed
width- width in Hz of the frequency bands

Curve

Multiplies spectrum magnitudes by an exponential curve ranging in amplitude from 1 to 0. This can be used for simple low- and high-pass filtering with adjustable roll-off amount.

Parameters:

freq- starting frequency of the curve

Class	**Name**	**Other Names**
Magnitude	Curve	High-/Low-pass Filter
	Impulse	Harmonize
	Moving Average	
	Offset	Noise Reduce
	Pass	Threshold, Noise Reduce, Gate
	Reject	Suppress, Limit
	Saw	
	Scale	Gain, Equalize (EQ), Filter
	Thin	Disintegrate/Coalesce, Scatter
	Thin Morph	
	Triangle	Feedforward Comb Filter
Frequency	Biscale	
	Offset	Frequency-shift
	Pack	
	Reverse	Invert
	Scale	Pitch-shift
	Scatter	Shake
	Swap	
	Warp	Frequency-stretch
Phase	Offset	
	Scale	
Time	Freeze	
	Offset	Delay
	Scale	Time-stretch, Time-contract
Peak	Pass	Trace
	Reject	Suppress, Limit
	Voices	Bare
Utility	Range	Band
	Seed	

Figure 3.4: Hierarchy of Transfer spectral processing modules and their relationship to known techniques.

width- frequency width of the curve in Hz where positive is low-pass and negative is high-pass
curve- degree of curvature

Impulse

Multiplies spectrum by a set of linearly spaced unit impulses. Has the effect of making a pitched sound at the frequency distance between the impulses. Harmonic pitches can be made with integer scaling ratios and inharmonic sounds can be created with non-integer scale values.

Parameters:
freq- frequency spacing between impulses
scale- amount to scale frequency spacing after first impulse

Saw

Multiplies spectrum magnitudes by a saw wave with amplitude ranging from 0 to 1. Has a similar effect as a time domain feedforward comb filter.

Parameters:

freq- frequency of the saw wave
phase- initial phase from 0 to 1 of the waveform starting at 0 Hz
direction- whether the wave ramps up or down
order- how many times to multiply the spectrum by the saw wave

Triangle

Multiplies spectrum magnitudes by a triangle wave with amplitude ranging from 0 to 1. Has a similar effect as a time domain feedforward comb filter.

Parameters:

freq- frequency of the triangle wave
phase- initial phase from 0 to 1 of the waveform starting at 0 Hz
order- how many times to multiply the spectrum by the triangle wave

Magnitude Average

Applies moving average filter to magnitudes. This tends to blur the spectrum resulting in a more noisy sound with a similar formant structure.

Parameters:

order- number of bins to average at a time

Magnitude Pass

Passes only bins above a certain magnitude threshold. This transformation is commonly used for noise reduction taking advantage of the fact that unwanted noisy parts of the spectrum tend to have lower mangitudes.

Parameters:

threshold- threshold value in dB
relativity- crossfade between absolute and relative thresholding

Magnitude Reject

Rejects bins above a certain magnitude threshold. This has the opposite effect of magnitude passing- only the noisy components of the spectrum are passed through.

Parameters:

threshold- threshold value in dB
relativity- crossfade between absolute and relative thresholding

Magnitude Scale/Offset

Scales and offsets spectrum magnitudes. This can be used for gain control or for simple spectral filtering by applying it within a range of frequencies. Offsetting the magnitudes by negative amounts has a similar effect to magnitude passing.

Parameters:
scale- amount to scale magnitudes
offset- amount in dB to offset magnitudes

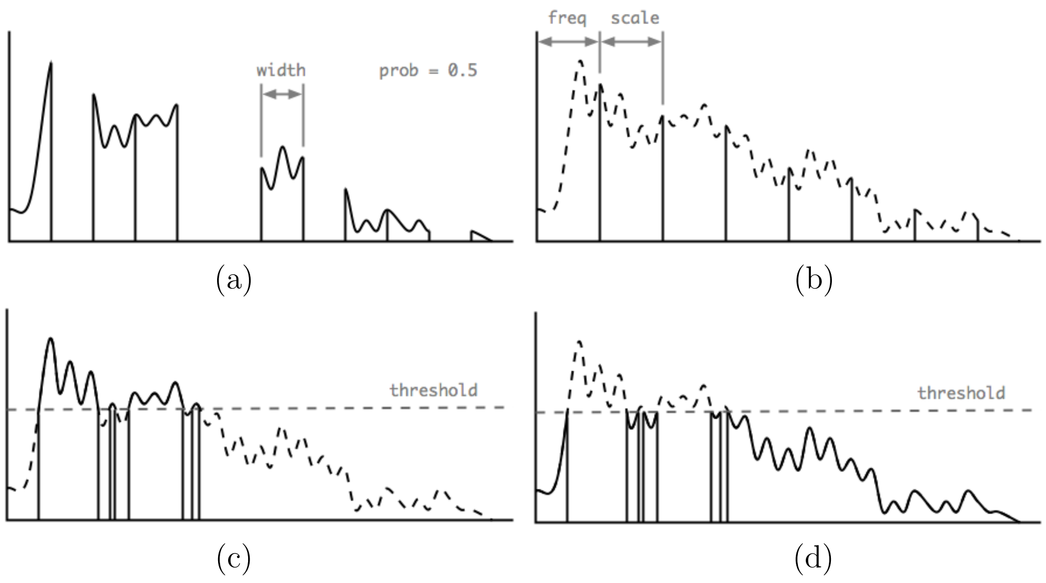


Figure 3.5: Magnitude transformations (a) band thin, (b) comb, (c) pass, and (d) reject.

3.3.2 Frequency Transforms

Frequency based transformations are desirable to do with the phase vocoder due to the intuitive nature of frequency samples and since it is very easy to avoid frequency aliasing. Time domain pitch-shifting and frequency-shifting introduce pronounced artifacts when shifting upwards by large amounts since high frequencies fold over at the Nyquist frequency. Also, time domain processing does not offer any control over the movement of bands of frequency content in the sound. Spectral domain frequency transformations are commonly used to change the pitch of a sound without effecting its duration.

Band Scatter

Scatter random-sized frequency bands around spectrum.

Parameters:
amount- maximum scatter amount of bands in Hz
widthMin- minimum width of band
widthMax- maximum width of band

Frequency Pack

Packs non-zero magnitude bins towards the low end of the spectrum.

Parameters:

none

Frequency Reverse

Reverses spectrum.

Parameters:

none

Frequency Scale/Offset

Scales and offsets spectrum frequencies. Scaling and offseting correlate to pitch-shifting and frequency-shifting.

Parameters:

mul- amount to scale frequencies
add- amount in Hz to offset frequencies

Frequency Warp

Warps frequencies around an inflection frequency.

Parameters:

freq- frequency to warp around
warp- degree of warping where 0 is none, negative is outward and positive is inward

3.3.3 Phase Transforms

Phase transformations are useful for accomplishing traditional time domain effects such as flanging, chorusing, and phasing. It is also possible to accomplish finer scale pitch-shifting and comb filtering by scaling and offsetting the phases.

Phase Scale/Offset

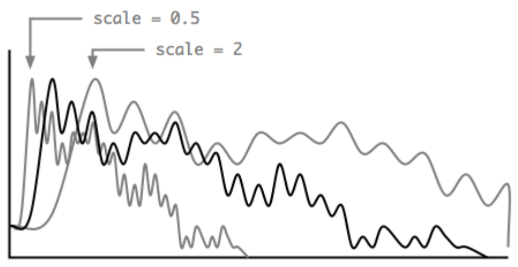
Scale and offset bin phases for a feedforward comb-filtering effect.

Parameters:

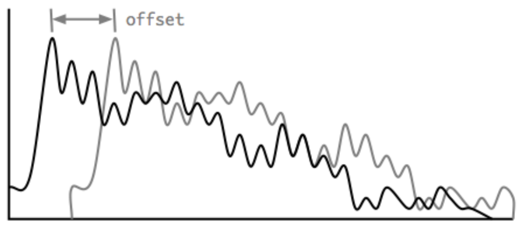
scale- amount to scale phases
offset- amount to offset phases

Phase Randomize

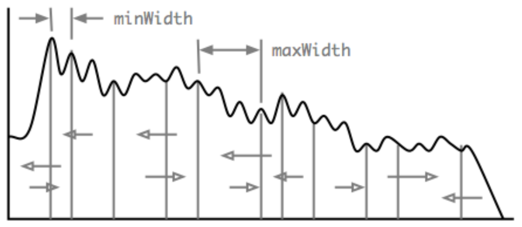
Randomize bin phases for chorus-like effects.



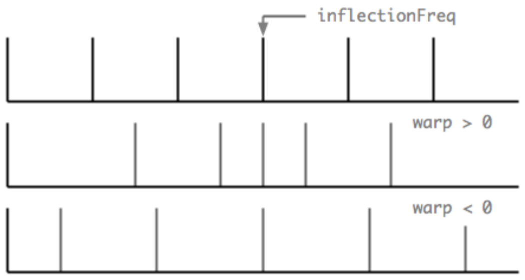
(a)



(b)



(c)



(d)

Figure 3.6: Frequency transformations (a) scale, (b) offset, (c) band scatter, and (d) warp.

Parameters:
amount- amount in radians to randomize phases

3.3.4 Time Transforms

The phase vocoder is unparalleled in terms of time-based transformations such as stretching and contracting. This is mainly due to the fact that the phases of the sinusoids can be accumulated from one frame to the next avoiding phase interference and modulation artifacts when overlapping windows. Spectral domain time transformations are typically used to alter the duration of sounds without effecting their pitch.

Time Freeze
Freeze time by keeping magnitudes constant.

Parameters:
freeze- whether or not to freeze the spectrum

Time Scale/Offset
Scale and offset time. Scaling and offseting correlate to time-stretching and time-delaying.

Parameters:
scale- amount to scale time

offset- amount to offset time

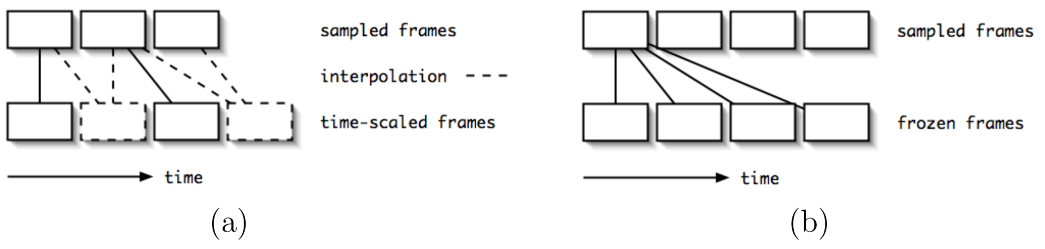


Figure 3.7: Time transformations (a) scale and (b) freeze.

3.3.5 Peak Transforms

These transformations operate on a higher level feature of the spectrum called a peak. Peaks are defined as bins whose magnitude is greater than its two neighbors. Peak detection is a commonly used technique for extracting the sinusoidal components from the rest of sound which is assumed to be some flavor of noise.

Peak Low Pass

Pass only the nth lowest frequency peaks.

Parameters:

peaks- number of peaks to pass
neighbors- how many neighboring bins to pass with peaks

Peak Pass

Pass only the nth highest magnitude peaks.

Parameters:

peaks- number of peaks to pass
neighbors- how many neighboring bins to pass with peaks

Peak Reject

Reject the nth highest magnitude peaks.

Parameters:

peaks- number of peaks to reject
neighbors- how many neighboring bins to pass with peaks

Voices

Attempts to pass only harmonically related peaks.

Parameters:

voices- number of voices to pass through
tolerance- maximum harmonic deviation for valid voice
harmonics- minimum number of harmonics for valid voice
neighbors- how many neighboring bins to pass with peaks

3.3.6 Utilities

For any of the above transformations it is possible to selected a range of frequencies to operate on. One example of its use is in creating simple spectral filters by magnitude scaling selected bands of frequencies. Another useful utility is a seeder for controlling random-based transformations, such as band thinning and band scattering.

Range

Specify a certain frequency range for spectral filters.

Parameters:

low- lower bound of range in Hz
high- upper bound of range in Hz
offset- amount to offset range in Hz

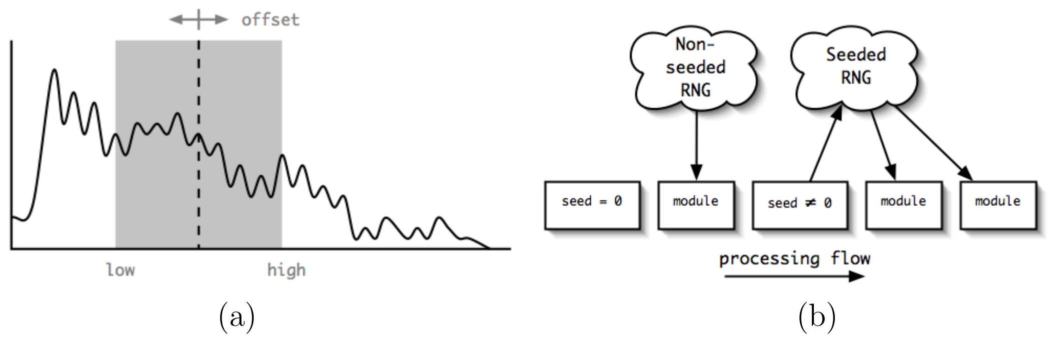


Figure 3.8: Utility modules (a) range and (b) seed.

Seed

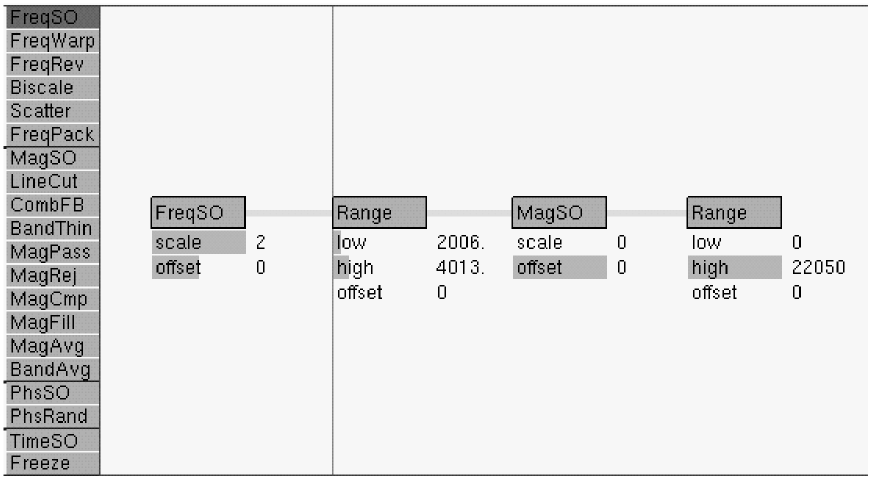
Plant a seed for the seeded random number generator (RNG). There are two RNGs used for all random-based transformations- non-seeded and seeded. The non-seeded RNG is used continuously from frame to frame while the seeded RNG is reseeded at least once every spectral frame. In other words, the non-seeded RNG applies extra-frame randomness to processing while the seeded RNG applies intra-frame randomness.

Parameters:

seed- seed for the local RNG, where 0 is a bypass

3.4 Module-level Interaction

The module screen is divided into two areas: the palette and the graph. The basic premise is that processing modules can be instantiated from a palette and moved around freely in a signal processing chain. The graph view follows the simple convention of signal flow from left to right. The graph allows two distinct types of processing flow: serial and parallel. A serial flow passes a transformed signal from one processor to another processor and a parallel flow splits a signal into two or more paths that are processed independently and summed together at some later point in time. By default, the processing flow is serial, since this is the most commonly used. The palette view contains selectable cells of the various types of transformations. Each row contains transformations of a specific class- magnitude, frequency, phase, time, peak and utility. The transformations can be placed in the processing graph by clicking its cell and pressing the spacebar. The transformation will be placed immediately after the last executing, or rightmost, module in the graph.



FreqSO
scale 2
offset 0

Range
low 2006.
high 4013.
offset 0

MagSO
scale 0
offset 0

Range
low 0
high 22050
offset 0

Figure 3.9: Screenshot of the Transfer module palette and graph editor.

Interaction with the modules on screen is done primarily with the mouse. Clicking and dragging a module with the left mouse button moves its position on screen. A simple cross-hair is shown at the top left corner to indicate the actively selected module and its position relative to other modules in the signal processing chain. Clicking and dragging an empty section of the workspace with the left mouse button moves all the modules as a group. Clicking with the right mouse button on a module will toggle the visibility of its parameter sliders. The module's parameters can then be modified by clicking and dragging the slider where the left button does course adjustment and the right button does fine adjustment.

3.5 Visual Feedback of Auditory Signals

Having the ability to visually represent sound is an important feedback device in any musical software. Not only does it allow one to perceive qualities of sound that would be difficult or impossible to hear, but its mode of feedback does not interfere with the listening of current auditory tasks. Showing the history and high frequency content of sound becomes an almost trivial task when done visually. Visual feedback in Transfer is presented through a scope with four different views: waveform, spectrum, wavescan, and sonogram. The waveform view displays an instantaneous time domain representation of the sound while the wavescan view shows a history. The spectrum and sonogram views are their analogues in the frequency domain.

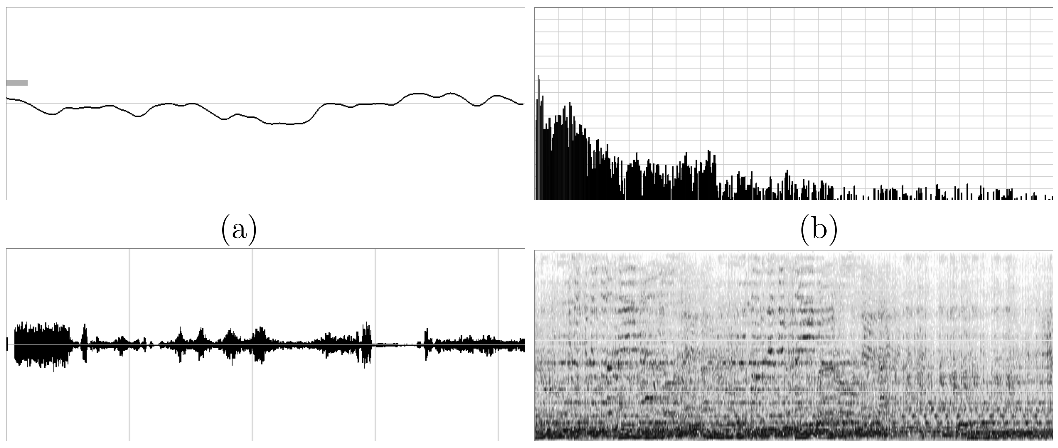


Figure 3.10: Screenshots of the four separate scope modes (a) waveform, (b) spectrum, (c) wavescan, and (d) sonogram.

The views have additional features that reveal important features of the sound. To help see finer details of the sound information each view has the ability to zoom in or out. In time domain views this simply controls the scale of the time-axis, while in the frequency domain it controls the number of frequency bins from 0 Hz that are displayed. The wavescan and sonogram views also lay down vertical bars at quarter-second marks as they scan across the screen from left to right.

Chapter 4

Transfer Implementation

Transfer was written in the C and C++ programming languages and uses several cross-platform libraries for low-level system operations. It uses OpenGL for its graphics rendering, PortAudio for generating audio and a custom C library for doing spectral processing. It has been tested to run under Mac OS X and could easily be ported to Windows since it uses all cross-platform APIs.

Table 4.1: Third-party libraries used by Transfer

Name	Used For
FFTW	Fast Fourier transform
GLUT	Windowing for OpenGL, keyboard/mouse handling
libsndfile	Time domain sound file I/O
OpenGL	Graphics rendering
PortAudio	Audio I/O

Transfer's program design is based and the Model-View-Controller (MVC) paradigm. The first component of Transfer are the models which are responsible for how data is processed. The primary model, the PVUnit, is an abstraction for the phase vocoder which does short-time analysis and resynthesis of time domain data and stores relevant information such as analysis parameters and time and spectral buffers. The second model, the Module, is an object that processes spectral data and stores information such as the transformation name, the parameter names, the parameter values, and the parameter ranges. The third model, the Graph, holds onto a group of Modules and controls the flow of spectral data through them. It has methods for inserting, removing, and moving Modules in the signal graph and a single process method that recursively calls each Module's process method in the proper order. Lastly is the Palette, a collection of all the available processing modules from which new Modules are inserted into the Graph object. The second design component of Transfer are the views which determine how the models are displayed graphically. At the core of each view is a rectangle with x-y position, width and height, background color, and border thickness and color. The views simply store a pointer to a model and have a draw method for displaying the data. The final element of MVC used in Transfer is the controller which is the user's interface to the views and/or models. The controllers map mouse and keyboard input onto certain functions that modify the state of views which in turn may modify their underlying models. Figure 4.1 shows a diagram of the relationships between the models and views within Transfer.

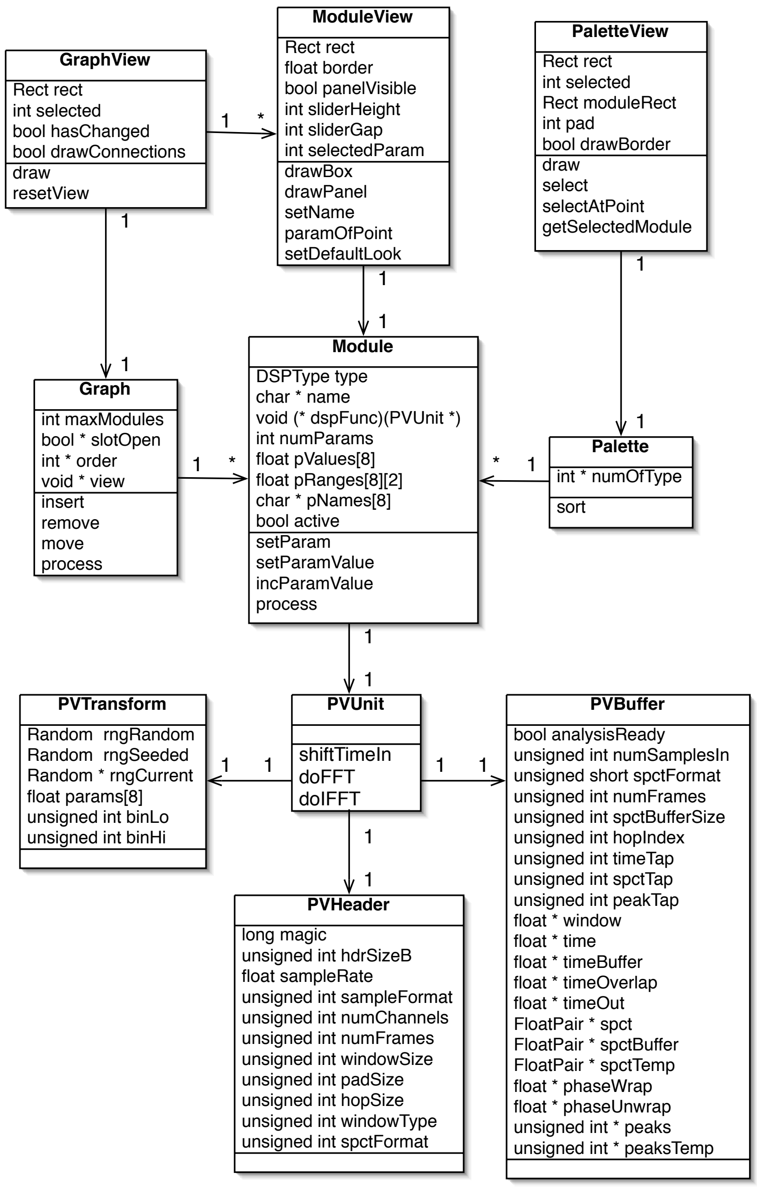


Figure 4.1: UML diagram of Transfer's models and views.

Chapter 5

Results

One thing that became evident during the implementation of Transfer was that it became much easier to debug spectral algorithms due to real-time audio/visual feedback and interaction. In the early stages of developing the spectral processing library, it was difficult and time-consuming to test spectral processing modules since the code had to be recompiled for each new test and fancy algorithms were required to test dynamic parameter changes. It also become more and more confusing to navigate the code when attempting to test chains of modules. Through implementing a graphical interface, the module testing process became much faster due to having a module graph editor and mouse control of parameter values. Also, previously unheard bugs in the processing algorithms became visually evident through the scope.

Another strength of Transfer was that it required very little setup time to create signal processing graphs. Although not extremely flexible, it was nice to be able to drag modules on the screen to determine their position in the processing graph. Also, it was quick and easy to gain access to module parameters through a single mouse click and have control over them with the mouse. Many existing module based programs require the user to do things such as write source code and/or manually connect control sliders to parameters all in an effort to do simple, common tasks. There is a clear benefit to the user if the most commonly done tasks are also the easiest to accomplish.

With respect to the quality of the frequency domain sound transformations in Transfer, it was found that they had less abrasive and noticeable artifacts than many time domain effects. Due to the fact that the phase vocoder uses overlapping windows during resynthesis, there were no undesirable pops or clicks introduced by the processing algorithms and by rearrangement of the module graph. This could be advantageous for a live performance that requires switching between different preset arrangements of modules.

Chapter 6

Future Work

Although a single piece of software can be extended to do a wide array of tasks, there are several important features that could be added to Transfer to improve its usability. An obvious one is the ability to store and recall your workspace to disk so module interconnections do not have to be done from scratch every time the program is restarted. Also, within the workspace it would be nice to store and recall preset configurations of module parameters. Another feature would be to extend the breadth of interaction to include sample-level editing and sequence-level event scheduling for compositional purposes. An interesting interface for a sequencer would be a table based view similar to a Csound score, but traversable like a spreadsheet. This could be extended further to allow encapsulating events within events and having code as events for doing more complex flow.

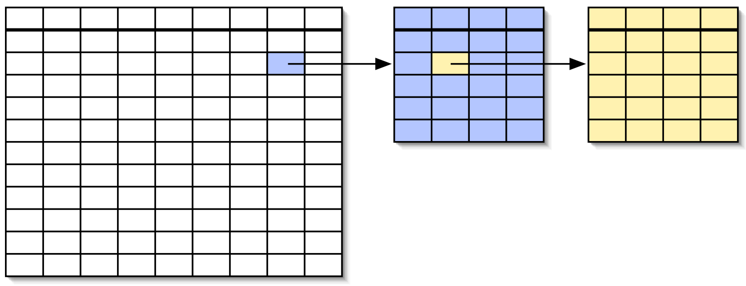
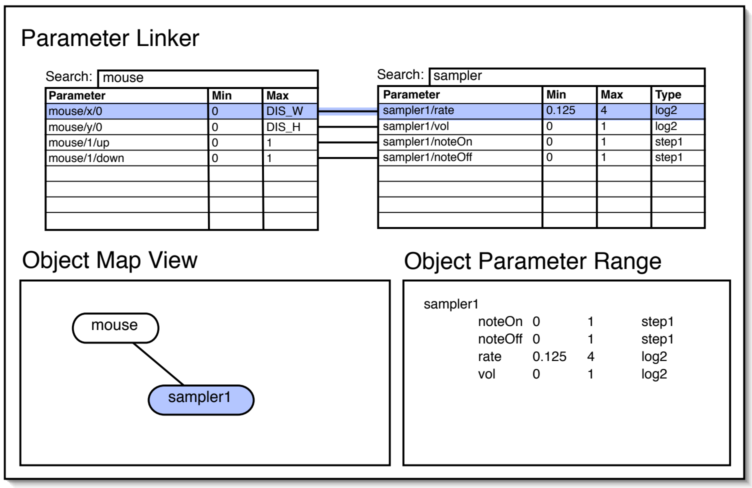


Figure 6.1: Sketch of a table-oriented interface for scheduling nested events.

Musically speaking, the module-level form of interaction is generally only useful for live performance. To extend the module-level interaction further it would be useful to have a general purpose input device to parameter mapper. A sketch of a possible interface is shown in Figure 6.2. The mapper is based on a querying mechanism to navigate all the recognized input device parameters and all the software's mappable parameters. The range of each parameter's value and type of mapping curve could also be specified in a table-oriented view. It would also be helpful to navigate mappable devices through a graphical map view that shows all the mapping connections and parameter details of clicked upon objects.


Parameter Linker

Search: mouse

Parameter	Min	Max
mouse/x/0	0	DIS_W
mouse/y/0	0	DIS_H
mouse/1/up	0	1
mouse/1/down	0	1

Search: sampler

Parameter	Min	Max	Type
sampler1/rate	0.125	4	log2
sampler1/vol	0	1	log2
sampler1/noteOn	0	1	step1
sampler1/noteOff	0	1	step1

Object Map View

Object Parameter Range

```
sampler1
         noteOn   0       1       step1
         noteOff  0       1       step1
         rate     0.125   4       log2
         vol      0       1       log2
```

Figure 6.2: Sketch of a parameter mapping screen.

Chapter 7

Conclusion

The frequency domain grants musicians the ability to work separately with the time and frequency content of sound. Today's computers allow us to operate on sound in the spectral domain with real-time visual feedback and interactive control. Unfortunately, modifying spectral data in an intuitive manner remains a problem. Transfer is a software program that allows one to control processes in the spectral domain through a graphical modular interface. Complex transformations can be accomplished by chaining together processing modules divided into six different types: magnitude, frequency, phase, peak, time, and utility. The graphical nature of Transfer makes it easy to control the parameters of spectral modules, see and modify the processing flow of modules, and to visualize the sonic effects of the transformations. This real-time interaction led to more rapid experimentation and evaluation of different spectral processing algorithms and configurations of modules. Transfer's model of intuitive, interactive, graphical control shows much promise for a future where an ever increasing amount of complex synthesis methods can be executed in real-time.

Appendix A

Fourier Transform

The Fourier transform, named after French mathematician and physicist Jean Baptiste Joseph Fourier, is a method for representing any arbitrary signal as a sum of sinusoids with specific amplitudes and phases. The sinusoids of the Fourier Transform are called its basis functions. There are four classes of the Fourier Transform derived from the nature of the signal being analyzed. For clarity, illustrations of the four signal types are shown in figure A.1.

Table A.1: Family of Fourier Transforms

Transform	Signal (Time)	Basis (Frequency)
Continuous Fourier Transform	Continuous, Aperiodic	Continuous, Aperiodic
Fourier Series	Continuous, Periodic	Discrete, Periodic
Discrete Time Fourier Transform	Discrete, Aperiodic	Continuous, Aperiodic
Discrete Fourier Transform	Discrete, Periodic	Discrete, Periodic

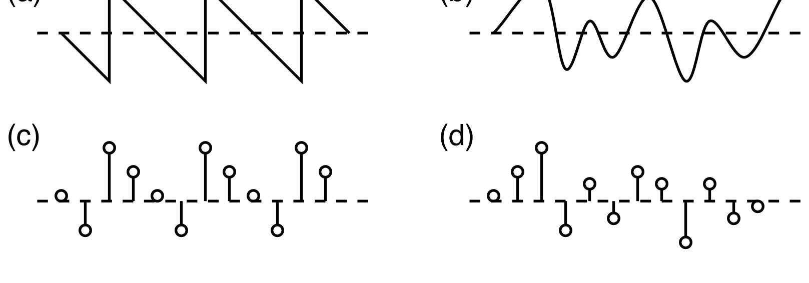


Figure A.1: Signals that are (a) continuous and periodic, (b) continuous and aperiodic, (c) discrete and periodic, and (d) discrete and aperiodic.

In the interest of presenting information relevant to the phase vocoder, only the discrete Fourier transform will be discussed. Before going any further, the following table will be presented to describe the most common symbols used.

Table A.2: Symbol definitions

Symbol	Description	Unit
x	Time domain frame	-
N, N_{DFT}	Number of samples in time-frame	samples
n	n^{th} sample in time-frame	-
X	Frequency domain frame	various
K	Number of bins in frequency-frame	bins
k	k^{th} bin in frequency-frame	-
R_t	Time domain sampling rate	samples/sec
R_f	DFT analysis rate	samples/sec
F_f	Fundamental bin frequency	Hz
$\omega(k)$	Angular frequency of k^{th} bin	radians/sample
$f(k)$	Frequency of k^{th} bin	Hz
$m[k]$	Magnitude of k^{th} bin	-
$\theta[k]$	Phase of k^{th} bin	radians
$I[k]$	Instantaneous frequency of k^{th} bin	Hz
N_{hop}	Number of samples between DFTs	samples
N_{pad}	Number of samples of zero-padding	samples
N_{win}	Number of samples in window	samples

A.1 Discrete Fourier Transform

The discrete Fourier transform (DFT), decomposes a discrete signal into a harmonically related, finite set of sinusoids. It is common to refer to the time domain signal as x and its frequency domain representation as X. N is the number of samples in the input signal operated on by the DFT and K is the number of frequency samples, referred to as bins, in the resulting analysis. $x[n]$ is the amplitude of the n^{th} sample in the time-signal and $X[k]$ is the magnitude and phase of the k^{th} harmonic in the frequency spectrum. Equations (A.1) and (A.2), respectively, are the mathematical descriptions for transferring to and from the frequency and time domains.

$$X[k] = \sum_{n=0}^{N-1} x[n](\cos(\omega(k)n) - j\sin(\omega(k)n)) \tag{A.1}$$

$$x[n] = \frac{1}{N}\sum_{k=0}^{K-1} X[k](\cos(\omega(k)n) + j\sin(\omega(k)n)) \tag{A.2}$$

$\omega(k)$ is the angular frequency of the k^{th} bin derived from

$$\omega(k) = \frac{2\pi k}{N}. \tag{A.3}$$

Just as time domain samples represent the amplitude of a waveform at discrete points in time, frequency domain samples represent the amplitude and phase of sinusoids at discrete points in frequency. The frequency of the analysis bins are related to the sampling rate, R_t, of the input signal and the size of the DFT, N_{DFT}. The fundamental frequency of the analysis, i.e. the first non-DC bin, is computed as:

$$F_f = \frac{R_t}{N} \tag{A.4}$$

From this, the frequency of the k^{th} bin can be calculated.

$$f(k) = kF_f \tag{A.5}$$

Equation (A.4) is simple, yet profound in that it shows that the fundamental frequency of the Fourier analysis is inversely proportional to the number of time samples being analyzed. In other words, there is a trade off between time resolution and frequency resolution. This makes sense since the period of a waveform is inversely proportional to its frequency. Due to the mechanics of the DFT, the lowest frequency sinusoid that can be detected in a signal is one with a period of N.

A.2 Data Representations

A.2.1 Samples

Before diving into specific representations for spectral data it is useful to compare the format of samples in both the time and frequency domains. In the time domain, samples are usually represented as a contiguous 1-D array of amplitude values. The independent variable time, is specified as a value in seconds between samples. In the frequency domain, samples are represented as a 2 x M matrix of sinusoidal data. A single frequency sample, or bin, requires two data points to fully describe a sinusoid. The independent variable frequency, is specified as a value in Hz between bins.

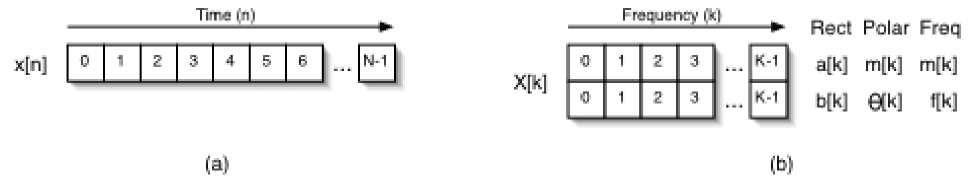


Figure A.2: Samples in the (a) time domain and (b) frequency domain.

For real to complex DFTs, the relationship between number of time domain samples, N, and frequency domain samples, K, is as follows:

$$K = \frac{N}{2} + 1. \tag{A.6}$$

Looking at (A.6), it may seem like we have created information since we went from N values in the time domain to $N + 2$ values in the frequency domain. This is truely not the case and lies with the fact that the components $X[0]$ and $X[K-1]$, the DC and Nyquist bins, have no phase information.

$$\theta_{DC} = \theta[0] = 0 \tag{A.7}$$

$$\theta_{Nyquist} = \theta[K-1] = 0 \tag{A.8}$$

Frequency domain samples have several formats for how sinusoidal data is stored in the bins. There is no one format that is best in all situations, therefore multiple representations are usually used simultaneously when processing spectral data.

A.2.2 Rectangular Form

The result of doing a DFT on an array of N real value samples, x, is an array of K complex value pairs, $(a[k],b[k])$, where

$$X[k] = a[k]\cos(\omega(k)n) + b[k]\sin(\omega(k)n). \tag{A.9}$$

The complex value pairs are the coordinates of a phasor on the complex plane.

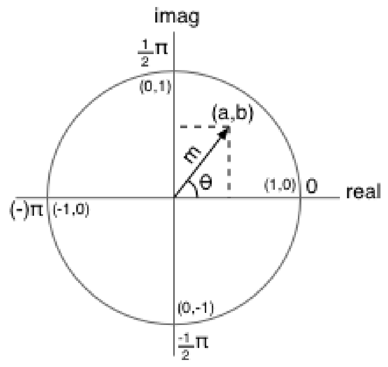


Figure A.3: The complex plane and unit circle.

Rectangular form is generally used for multiplying spectra together to do convolution in the time domain.

A.2.3 Polar Form

A more intuitive data representation for frequency samples is to specify them as the magnitude and phase of a sinusoid. $X[k]$ can be represented in polar form as

$$X[k] = m[k]\sin(\omega(k)n + \theta[k]) \tag{A.10}$$

by converting it from its rectangular form (A.9) using:

$$m[k] = \sqrt{a[k]^2 + b[k]^2} \tag{A.11}$$

$$\theta[k] = \tan^{-1} \frac{a[k]}{b[k]} \tag{A.12}$$

To do the inverse DFT, the frequency samples must be put back in rectangular form using:

$$a[k] = m[k] \cos(\theta[k]) \tag{A.13}$$

$$b[k] = m[k] \sin(\theta[k]) \tag{A.14}$$

Polar form is most useful for frequency analyzers, spectral feature extraction, and as an intermediary format for converting to magnitude/frequency form.

A.2.4 Magnitude/Frequency Form

Besides having the magnitude and phase of a sinusoid in a bin, it is informative to have an estimate of the sinusoid's true frequency. Remember that frequency samples are quantized, so partials that don't lie on integer multiples of F_f will have energy spread across multiple bins. Fortunately, we can make a good guess of a partial's actual frequency by using phase information. The frequencies in this form are called instantaneous frequencies and they are obtained by taking the time derivative of phase values between successive analysis frames. The instantaneous frequency of the k^{th} bin is

$$I[k] = \frac{R_t}{2\pi N_{hop}} wrap_{\{-pi,pi\}}(\theta'[k] - k\theta_f) + kF_f \tag{A.15}$$

where $\theta'[k]$ is the time derivative of the k^{th} bin's phase between analysis frames and θ_f is the number of radians we expect to go by each N_{hop} samples for a sinusoid at the fundamental analysis frequency, F_f. θ_f, the fundamental radian increment, is defined as:

$$\theta_f = \frac{2\pi N_{hop}}{N} \tag{A.16}$$

Converting back to polar form is done simply by solving equation (A.15) for $\theta'[k]$ and accumulating the phase difference.

$$\theta'[k] = \frac{2\pi N_{hop}}{R_t}(I[k] - kF_f) + k\theta_f \tag{A.17}$$

Equation (A.17) can be greatly simplified by substituting in equations (A.4) and (A.16) for F_f and θ_f, respectively.

$$\theta'[k] = \frac{2\pi N_{hop}}{R_t}(I[k] - k\frac{R_t}{N}) + k\frac{2\pi N_{hop}}{N}$$

$$\theta'[k] = \frac{2\pi N_{hop}}{R_t} I[k] - k\frac{2\pi N_{hop}}{N} + k\frac{2\pi N_{hop}}{N}$$

$$\theta'[k] = \frac{2\pi N_{hop}}{R_t} I[k] \tag{A.18}$$

Magnitude/frequency form is used for pitch- and time-scaling, oscillator bank resynthesis, and as a starting point for more advanced analysis techniques such as peak detection and continuation.

Appendix B

STFT Analysis Parameters

In general, there are four analysis parameters that need to be specified before analysis can take place. These are the sizes (in samples) of the window, zero-padding, and analysis hop, and the type of window. Many other combinations of these parameters can be derived and used as specifications depending on taste. Some of the more popular ones are DFT size (window size plus zero-padding size), analysis rate (sample rate divided by hop size), and overlap factor (window size divided by hop size). Some window types may also have one or more additional parameters to specify their shape.

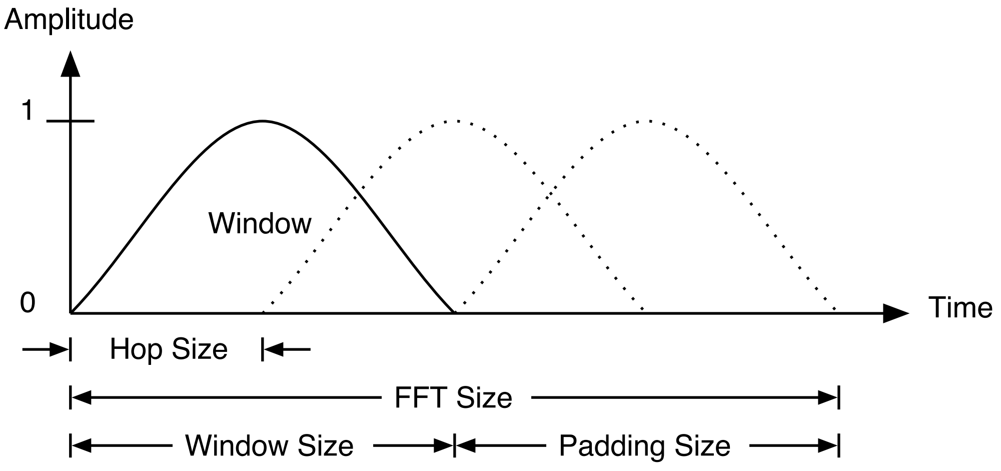


Figure B.1: Analysis parameters

The choice of analysis parameters will have a large impact on the nature of the analysis data, including frequency resolution, amount of data, and smearing artifacts. The amount of each analysis parameter will have a direct or inverse relationship with certain qualities of the spectral data.

The choice of window shape will have a large impact on the type of smearing of the frequency samples. Technically speaking, this smearing is convolution in the frequency domain of the spectra of the window and samples since they are being multiplied in the time domain.

Table B.1: Analysis Parameters

Parameter	Frequency Resolution	Time Resolution	Typical Ranges
N_{win}	direct	inverse	[512, 4096]
N_{pad}	direct	-	$[0, 3N_{win}]$
N_{hop}	-	inverse	$[N_{win}/2, N_{win}/8]$

Table B.2: Characteristics of various window types

Type	Main-Lobe Width (-3dB) (bins)	First Side-Lobe Height (dB)	Roll-off (dB/octave)
Rectangle	0.89	-13	-6
Triangle	1.28	-27	-12
Hann	1.20	-31	-18
Hamming	1.30	-43	-6
Blackmann	1.68	-58	-18
Kaiser	variable	variable	-6

Ideally, we want a window that gives a narrow main lobe width, low first side lobe height, and high roll-off factor to minimize the amount of smearing artifacts. Unfortunately, in reality increasing the desirability of one trait tends to make the others worse. The most popular choices of windows are the Hann and Hamming since they offer a good balance between all the characteristics. Another popular window choice is the Kaiser window because it can be modified to give different trade-offs depending on the nature of the signal being analyzed.

Bibliography

[1] De Goetzen, A., Bernardini, N., and Arfib, D. “Traditional (?) implementations of a phase-vocoder: The tricks of the trade.” Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFx-00). Verona, Italy. 7-43.

[2] Dobson, R. 1993. “The operation of the phase vocoder: A non-mathematical introduction to the fast Fourier transform.” Composer’s Desktop Project. http://www.bath.ac.uk/ masjpf/CDP/operpvoc.htm

[3] Dolson, M. 1986. “The phase vocoder: a tutorial.” Computer Music Journal. 10(4):14-27.

[4] Flanagan, J. L. and Golden, R. M. 1966. “Phase vocoder.” Bell System Technical Journal. 1493-1509.

[5] Harris, F. 1978. ”On the use of windows for harmonic analysis with the discrete Fourier transform.” Proceedings of the IEEE. 66(1):51-83.

[6] Laroche, J. and M. Dolson. 1997. “Phase-vocoder: About this phasiness business.” Proceedings of the International Computer Music Conference.

[7] Laroche, J. and M. Dolson. 1999. “New phase-vocoder techniques for pitch-shifting, harmonizing and other exotic effects.” Proceedings of 1999 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics. New Paltz, NY.

[8] Manolescu, D. 1997. “A data flow pattern language.” Proceedings of the 4th Pattern Languages of Programming, Monticello, Illinois.

[9] McCartney, J. 2002. “Rethinking the Computer Music Language: SuperCollider.” Computer Music Journal. 26(4): 61-68.

[10] Moore, F.R. 1990. Elements of Computer Music. Englewood Cliffs: Prentice Hall.

[11] Moorer, J.A. 1978. “The use of the phase vocoder in computer music applications.” Journal of Audio Engineering Society. 27(3): 134-140.

[12] Nutall, A. H. 1981. ”Some windows with very good sidelobe behavior.” IEEE Transactions on Acoustics, Speech and Signal Processing. ASSP-29(1):84-91.

[13] Portnoff, M.R. 1976. "Implementation of the digital phase vocoder using the fast Fourier transform." IEEE Transactions on Acoustics, Speech and Signal Processing. ASSP-24(3):243-248.

[14] Puckette, M. 2002. "Max at seventeen." Computer Music Journal. 26(4): 31-43.

[15] Roads, C. 2001. Microsound. Massachusetts Institute of Technology.

[16] Serra, M.-H. 1997. "Introducing the phase vocoder." In Musical Signal Processing, ed. Curtis Roads et al, 31-90. Lisse: Swets & Zeitlinger.

[17] Wishart, T. 1994. Audible Design. Orpheus the Pantomime Ltd.

[18] Wishart, T. "Computer Sound Transformation." http://www.trevorwishart.co.uk/transformation.html (Accessed November, 2005).