institut für elektronische musik und akustik

IEM Report 34/06

Morphing zweier Klänge mit quasi-harmonischen Spektren

Verfasser:

Thomas Musil

Robert Höldrich

Oktober 2006

A-8010 Graz, Inffeldgasse 10/3, Tel.:+43/(0)316/389 – 3170, FAX:+43/(0)316/389 – 3171

office@iem.at http://iem.at

Zusammenfassung

Dieser Bericht befasst sich mit dem Thema Morphing zweier Klänge. Voraussetzung dazu ist, dass beide Klänge die Eigenschaft besitzen, sich mittels Partialtonsynthese darstellen zu lassen. Die Verbesserung zu früheren Versionen sind: 1.) Analyse und Morphing-Resynthese arbeiten auf dem selben Rechner, man benötigt nur handelsübliche Rechner-Hardware und freie Software (Pd); 2.) die Klanganalyse-Datenspeicherung verwendet das SDIF-Format (von IRCAM entwickeltes Format), damit herrscht auch Kompatibilität zu SVP- bzw. AudioSculpt-Analysen; 3.) verbesserte Aufbereitung der Analyse-Dateien mittels statistischer Filter. Zusätzlich notwendig war die Entwicklung von zwei Bibliotheken (iem_morph und morph_abs) für Pd , einer graphischen Echtzeit-Multimedia-Programmiersprache von Miller Puckette[1].

Abstract

This report deals with the topic of morphing of two sounds (provided that both sounds are able to be additive synthesized). The improvements of this new version are: 1.) analysis and morphing resynthesis are working on standard hardware and with free software (Pd); 2.) the results of sound analysis are stored in SDIF format.(developed by IRCAM), so there is a compatibility to other audio software like AudioSculpt and SVP; 3.) quality-enhancing by using statistical filters for data preparation. Two Pd libraries (*iem_morph* and *iem_abs*) are required to realize those goals in Pd (a realtime multimedia graphical programing language, developed by Miller Puckette [1]).

INHALTSVERZEICHNIS

1 Einleitung

Die Ideen zu den Verbesserungen vom Morphen zweier Partialtonklänge mit quai-harmonischen Spektren stammen zum größten Teil von Prof. Dr. Höldrich. Die Inspiration zur Partialtonanalyse stammt zum größten Teil von Miller Puckette und dessen pd-Objekt *fiddle~*, bei der Ausführung der technischen Details war Robert Höldrich sehr hilfreich. Die Bibliothek selbst besteht aus einer External-Bibliothek (iem_morph) mit den Endungen *.dll für Windows, *.pd_linux für Linux und einem Sammelordner morph_abs mit Pd-Subpatches.

Die aktuelle Version von iem_morph und morph_abs liegen als Daten-CD diesem Report bei.

2 Beschreibung der Partialtonanalyse

Voraussetzung dafür ist, dass der zu untersuchende Klang größtenteils eine annähernd harmonische Teiltonstruktur besitzt. Die Partialton-Analyse erfolgt dann in 2 Schritten:

2.1 Fundamentalton-Schätzung und Glättung

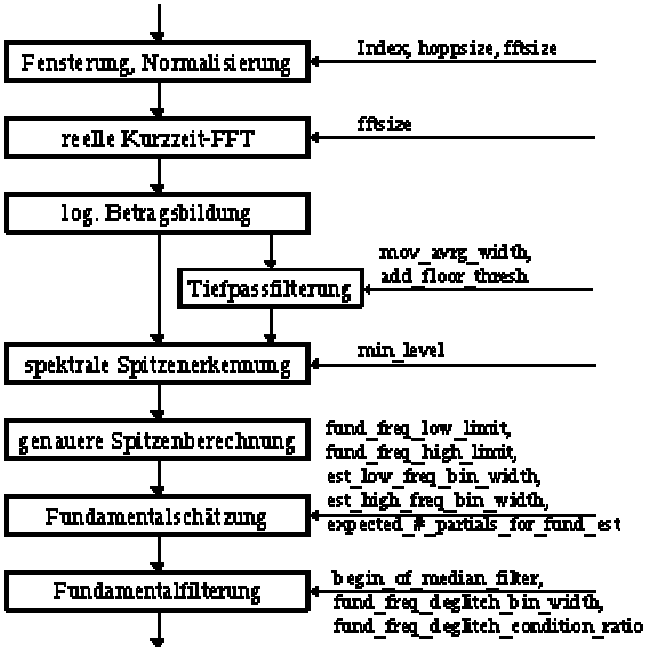

Abb. 1

Abb. 1 zeigt uns den Ablauf der Fundamentalton-Schätzung. Die hier verwendeten Parameternamen werden auch in den Pd-Patches werwendet. Zuerst entnehmen wir der Quelldatei *src_wav_name* einen Sample-Block der Größe *fftsize*, fenstern ihn mit einem Hannig-Fenster, normalisieren noch im Zeitbereich mit Faktor 1/(fftsize/4), führen diesen Signalblock einer realen FFT zu, quadrieren Real- und Imaginär-Teil, addieren beide Komponenten zur spektralen Leistungsdichte dieses Signalblocks, logarithmieren sie und speichern sie in Table *mag*, falten bzw. tiefpassfiltern sie mit einem normalisierten Hanningfenster der Breite *mov_avrg_width* und speichern sie in Table *floor*. Aus den Tables *mag* und *floor*, den Parametern Minimumpegel *min_level* und Noisefloor-Abstand *add_floor_thresh* wird eine Liste relativer Maxima gewonnen

$$mag[i-1] \le mag[i] > mag[i+1] \quad \vee \quad mag[i-1] < mag[i] \ge mag[i+1]$$

unter Berücksichtigung der beiden Zusatzbedingungen.

$$mag[i] > \min\_level \quad \wedge \quad mag[i] > floor + add\_floor\_thresh$$

Aus dieser Liste von relevanten spektralen Peaks werden nun die genaueren Frequenzen und logarithmischen Amplituden mittels einer 3-Punkt-Interpolation ermittelt. Der nächste Schritt ist nun eine Maximum Likelihood Estimation bezüglich der Wahrscheinlichkeitsverteilung der Partialton-Peaks im Spektrum (Noll [2]), welche sich harmonisch zur Fundamentalfrequenz aufbauen sollten. Wir verwenden dazu eine vereinfachte Likelihood-Funktion

$$L(f) = \sum_{n=1}^{\max} e^{-(f_{pk} - n * f_{fe})^2 / C} \quad .$$

n ist die Laufvariable der Summe, *max* ist die Anzahl der tieferen Teiltöne für die Fundamentaltonschätzung *number_of_partials_for_fund_est*, f_{pk} ist die zu gewichtende Peak-Frequenz, f_{fe} ist die geschätzte Fundamentalfrequenz, *C* berechnet sich aus der Bandbreite der Gauß-Verteilungsfunktion (*est_low_freq_bin_width* bzw. *est_high_freq_bin_width*) die sich auf die beiden Bandgrenzen von 0.5 beziehen. Die geschätzte Fundamentalfrequenz f_{fe} wird nun in erster Instanz zwischen den Grenzen *fund_freq_low_limit* und *fundfreq_high_limit* im Achtelton-Abstand variiert.

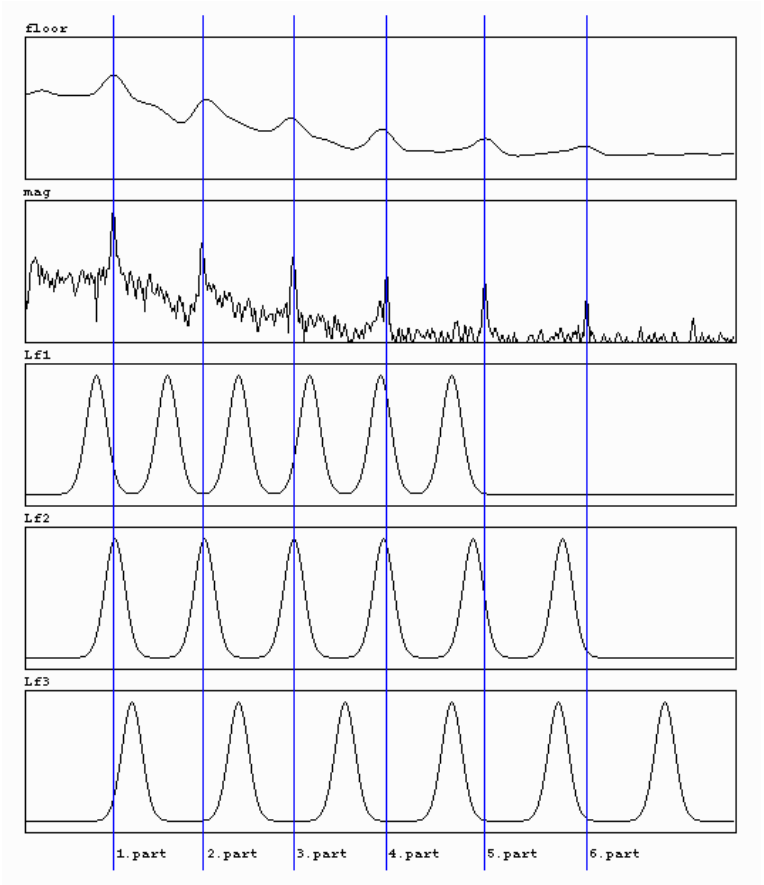

Abb. 2

L(f) soll dabei einen maximalen Wert erreichen. *Abb. 2* zeigt die 2 Tables *floor* und *mag*, darunter 3 Schätzwert-Verläufe *Lf1*, *Lf2* und *Lf3*. Die Fundamentalfrequenz bei

Lf1 wurde zu tief, bei *Lf3* zu hoch geschätzt. Im Falle von *Lf2* wurde die Fundamentalfrequenz ganz gut getroffen, man sieht jedoch, dass es sich beim Spektrum um einen Klavierton mit einer Inharmonizität der Teiltöne handelt.

Es folgen nun noch 4 weitere Durchläufe der Fundamentalton-Schätzung nach dem Prinzip der sukzessiven Approximation (die Differenz von Untergrenze und Obergrenze der Fundamentalfrequenz wird immer kleiner). Die zuletzt verwendete Frequenz wird nun in eine Table *fund_est* eingetragen.

Nachdem wir nun der ganzen Quelldatei im Abstand *hoppsize* die Datenblöcke entnommen und die Fundamentalfrequenz berechnet haben, filtern wir die Table *fund_est* mit einem bedingten Medianfilter. Wir beginnen im Punkt *begin_of_median_filter* (in Prozent bezogen auf Quelldateilänge) und bewegen uns zuerst Richtung Anfang und später Richtung Ende. Die Tiefe der Filterung wird mit dem Parameter *fund_freq_geglitch_bin_width* beschrieben, die Einsatz-Entscheidung durch den Parameter *fund_freq_deglitch_condition_ratio*. Zuerst wird ein Satz der Breite *fund_freq_geglitch_bin_width* der Table *fund_est* entnommen, bei rechtsläufiger Richtung ist der rechteste Wert der zu filternde Eingangswert x. Dann wird der Satz der Größe nach sortiert, der mittlere Eintrag der Liste ist der Medianwert y_m.

Zuletzt wird untersucht, ob $x > y_m$ * *fund_freq_deglitch_condition_ratio*, oder ob $x < y_m$ / *fund_freq_deglitch_condition_ratio* ist. Falls dies zutrifft, wird x als Glitch erkannt und durch y_m ersetzt.

2.2 Partialton-Berechnung und Speicherung

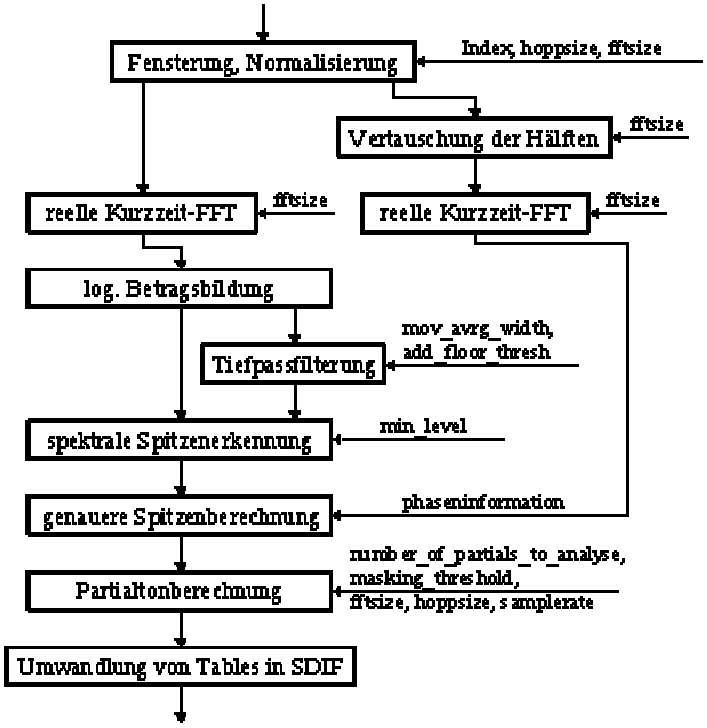

Abb. 3

Wir beginnen wieder wie vorher damit, dass wir zuerst der Quelldatei *src_wav_name* einen Sample-Block der Größe *fftsize* entnehmen und ihn (siehe *Abb. 3*) mit einem Hanning-Fenster multiplizieren, ihn noch im Zeitbereich mit dem Faktor 1/(fftsize/4) normalisieren. Hier spaltet sich der Signalweg auf, einerseits wird dieser Block in 2 Teile geteilt und diese vertauscht, einer reellen FFT zugeführt, deren Real- und Imaginär-Teil später die Phaseninformation der Peak-Frequenzen ergeben. Andererseits führen wir diesen Signalblock unbearbeitet einer realen FFT zu, quadrieren Realteil und Imaginärteil, addieren beide Komponenten zur spektralen Leistungsdichte dieses Signalblocks, logarithmieren sie und speichern sie in Table *mag*, falten bzw. tiefpassfiltern sie mit einem normalisierten Hanningfenster der Breite *mov_avrg_width* und speichern sie in Table *floor*. Wie bei der Fundamental-Schätzung werden aus den Tables *mag* und *floor*, den Parametern *min_level* und *add_floor_thresh* die relativen Maxima, gewonnen. Nun werden 3 Listen von genauer gerechneten Peak-Amplituden (linear), Peak-Frequenzen (in Hz) und Peak-Phasen ($-\pi$ .. $+\pi$) dieses Blocks mittels 3-Punkt-Interpolation erstellt. Der nächste Schritt verwendet die vorher ermittelte Fundamentalfrequenz, multipliziert sie mit 1 bis *number_of_partials_to_analyse* und erhält damit die zu erwartenden Partialtonfrequenzen. Nun sucht man im Bereich eines Teiltones

(+- 0.5*Fundamentalfrequenz) den stärksten Peak aus und tragt ihn in die jeweiligen Tables der Partialtonfrequenzen, Partialtonamplituden und Partialtonphasen ein. Falls noch ein zweiter Peak vorhanden ist, der größer als der erste Peak minus *masking_threshold* (in dB) ist, werden die beiden Amplituden linear addiert, die Frequenzen und Phasen proportional ihrer Amplitudenstärke linearkombiniert. Zuletzt werden diese 3 mal *number_of_partials_to_analyse* Tables noch als SDIF-Datei gespeichert. Diese besteht aus einem Header mit dem Erkennungsmerkmal „HRM“ für harmonisches Spektrum und darauf folgenden binär kodierten Matrixblöcken. Jeder analysierte Signalblock ergibt einen Matrixblock von 4 Spalten mal *number_of_partials_to_analyse* Zeilen. Die erste Spalte beinhaltet den Partialindex beginnend mit 1, in der zweiten Spalte steht die Partialfrequenz in Hz, in der dritten Spalte steht die lineare Partialamplitude und in der vierten Spalte die Partialphase ($-\pi$ .. $+\pi$).

3 Implementierung der Partialtonanalyse in Pd

Voraussetzungen: Rechner mit mehr als 1.2 GHz Pentium CPU, Windows oder Linux Betriebssystem, Pd-Version 0.39. Verwendete Externals: iemlib1, iemlib2, zexy, iem_tab, iem_spec2, iemgui, iem_morph. Verwendete Abstraktionssammlungen: iemabs, morph_abs.

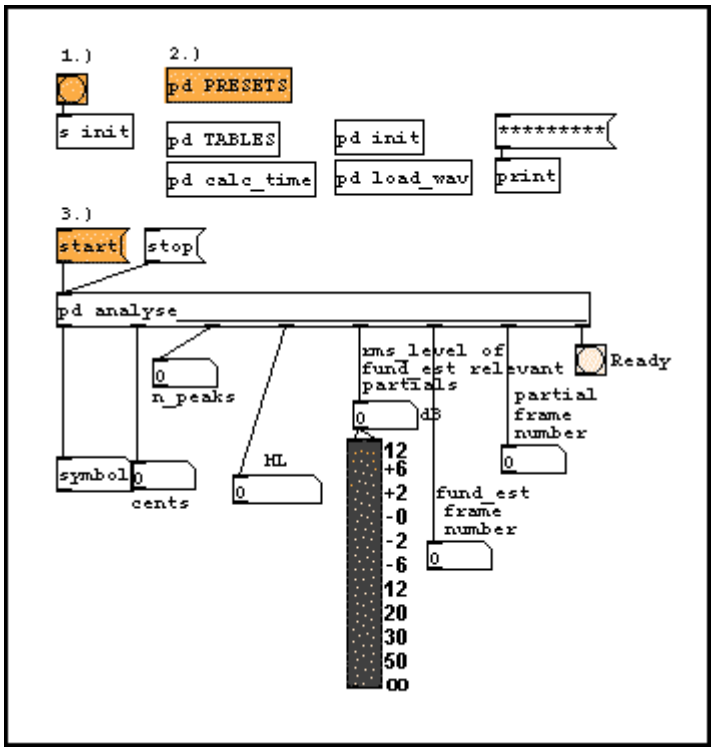

Abb. 4

Nach dem Starten von Pd öffnet man den Pd-Patch. *Partial_Analysis.pd* (siehe *Abb. 4*) und initialisiert 1.) grundlegende Größen wie: *fftsize*, *hoppsize* und *samplerate*. Dann öffnet man 2.) Subpatch PRESETS (siehe *Abb. 5*) und editiert in einer Messagebox die Parameter. 3.) startet man die Analyse, diese dauert bis zu 10 mal solange wie Echtzeit.

```
;
src_wav_name sounds/Sopran_a/a_k_g.wav;
add_floor_thresh 5;
mov_avrg_width 20;
est_low_freq_bin_width 1;
est_high_freq_bin_width 1;
fund_freq_low_limit G;
fund_freq_high_limit g1;
min_level -120;
number_of_partials_for_fund_est 16;
begin_of_median_filter 30;
fund_freq_deglitch_bin_width 55;
fund_freq_deglitch_condition_ratio 1.1;
masking_threshold 6;
number_of_partials_to_analyse 80;
dst_sdif_name sounds/Sopran_a/a_k_g.sdif
```

Abb. 5

Die Parameter bedeuten:

src_wav_name: Quelldateiname einer Mono-Wave-Datei;

add_floor_thresh: additiver Pegel zur geglätteten spektralen Leistungsdichte;

mov_avrg_width: Breite des hanning-förmigen gleitenden Mittelwert-Filters;

est_low_freq_bin_width: untere Bandbreite einer Gauß-Verteilung des Maximum Likelihood Estimators in FFT-bins;

est_high_freq_bin_width: obere Bandbreite einer Gauß-Verteilung des Maximum Likelihood Estimators in FFT-bins;

fund_freq_low_limit: unterste erwartete Fundamentalfrequenz in Noten;

fund_freq_high_limit: oberste erwartete Fundamentalfrequenz in Noten;

min_level: niedrigster Pegel eines spektralen Peaks in dB;

number_of_partials_for_fund_est: reduzierte Anzahl der Teiltöne für die Fundamentaltonschätzung;

begin_of_median_filter: Beginn der Störimpulsfilterung in der Speichertabelle für die Fundamentalfrequenz in Prozent;

fund_freq_deglitch_bin_width: Filterbreite der Störimpulsfilterung in der Speichertabelle für die Fundamentalfrequenz in FFT-bins;

fund_freq_deglitch_condition_ratio: Verhältnis von Eingangs- zu Median-Wert, stellt die Einsetzbedingung der Störimpulsfilterung in der Speichertabelle für die Fundamentalfrequenz dar;

masking_threshold: psychoakustischer Parameter in dB, entscheidet, ob ein schwächerer spektraler Peak innerhalb eines Teiltonbereichs berücksichtigt wird;

number_of_partials_to_analyse: maximale Anzahl der zu analysierenden Teiltöne;

dst_sdif_name: Zieldateiname zur Speicherung der Information der spektralen Teiltonverläufe.

Anzupassen sind hauptsächlich Quelldateiname, Zieldateiname und Fundamentalfrequenzbereich.

4 Implementierung der Partialtonresynthese in Pd

Der Pd-Patch *Partial_Synthesis_Cross_Stretch.pd* dient zum Beurteilen der Wahl der richtigen Parametergrößen bei der Partialtonanalyse. Die Resynthese basiert auf dem Modell der additiven Synthese von harmonischen Schwingungen. Eine genauere Beschreibung wird in den nächsten beiden Kapiteln erklärt. Die Voraussetzungen bezüglich Pd sind gleich wie beim vorherigen Analyse-Patch. Nach dem Starten von Pd-Patch *Partial_Synthesis_Cross_Stretch.pd* (siehe *Abb. 6*) schaltet man 1.) den DSP-Toggle ein, 2.) initialisiert samplerate und hoppsize, 3.) editiert und versendet die Preset-Messagebox mit den wichtigsten Parametern (siehe *Abb. 7*), 4.) editiert die Dehn-/Stauch-Table, 5.) filtert die Partialfrequenz-Tables, 6.) startet die Resynthese.

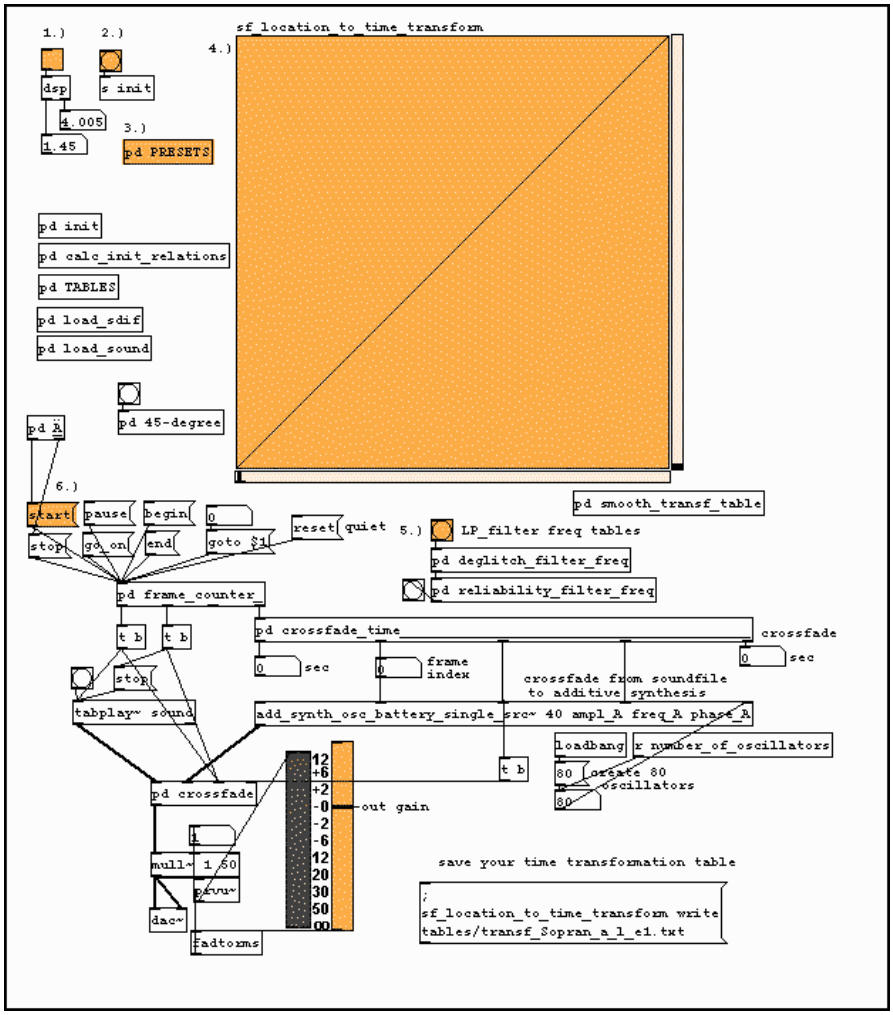

Abb. 6

```
;
number_of_partials_to_synthesize 80;
synthesis_crossfade 0.75;
stretch_factor 1;
partial_freq_LP_coeff 1;
number_of_oscillators 80;
deglitch_bin_width 55;
reliability_threshold -70;
src_sdif_name sounds/Klavier/D.sdif;
src_wav_name sounds/Klavier/D.wav;
sf_location_to_time_transform read
tables/transf_Klavier_D.txt
```

Abb. 7

Erklärung der Syntheseparameter:

number_of_partials_to_synthesize: selbe Anzahl der Teiltöne wie bei der Analyse;

synthesis_crossfade: Zeitpunkt in sec. der Überblendung von Klangabspielung zu additiver Synthese = Beginn der Störimpulsfilterung in den Partialtonfrequenz-Tables;

stretch_factor: Dehnungsfaktor der Abspielzeit;

partial_freq_LP_coeff: Tiefpasskoeffizient (0 .. 1) für Partialtonfrequenz-Glättungsfilter (Devibrato);

number_of_oscillators: reduzierte Anzahl der Oszillatoren;

deglitch_bin_width: Filterbreite der Störimpulsfilterung in den Partialtonfrequenz-Tables in bins;

fund_freq_deglitch_condition_ratio: Verhältnis von Eingangs- zu Median-Wert, stellt die Einsetzbedingung der Störimpulsfilterung in den Partialtonfrequenz-Tables dar;

reliability_threshold: Zuverlässigkeits-Pegel in dB, der entscheidet, dass Frequenzwerte mit logarithmischen Amplituden unter diesem Pegel nicht mehr vertrauenswürdig sind, und somit durch glaubwürdige Vorgängerwerte ersetzt werden;

src_sdif_name: Quelldateiname mit der Information der spektralen Teiltonverläufe;

src_wav_name: Quelldateiname der Mono-Wave-Datei;

sf_location_to_time_transform: ASCII-Quelldatei der Ort/Zeit-Transformations-Table zum Dehnen/Stauchen des Zeitverlaufs.

5 Beschreibung der Morphing-Synthese

Wir setzen voraus, es existieren jeweils die Partialton-Analysen zweier Klänge im SDIF-Format (Klang A und Klang B). Wir wünschen uns eine freiwählbare, veränderliche Mischung beider Klänge in Frequenz (unterschiedliche Grundtöne, Harmonizitäten, Vibrato) und Amplitude (unterschiedliche Formanten, Tremolos). Weiters stellen wir fest, dass wir interessantere Stellen länger hören wollen und dass die Klänge komplexe, nichtharmonische Anschlag- bzw. Ansatz-Geräusche besitzen. (siehe *Abb. 8*) Wir beginnen immer beim Abspielen des Originalklangs A, blenden über zum Klang A der additiven Synthese nach $t_{A\_crossfade}$ Sekunden und binnen hoppsize Samples (wichtig dabei ist, die exakte Phasenlage aller Oszillatoren zu Beginn der Überblendung zu stellen), dessen weiterer Verlauf ab diesen Punkt um *stretch_factor_A*-fach gedehnt wurde. Zum Zeitpunkt $t_{B\_start\_delay}$ wird die additive Synthese von Klang B gestartet, deren Länge um den *stretch_factor_B* mal gedehnt wurde. Im Zeitraum t_{Morph}, während dessen beide additiven Synthesen laufen, kann man die Partialtonfrequenzen und Partialtonamplituden frei zwischen Klang A und B hin und her blenden. Die beiden Tables „*soundfile location to time transformation table A/B*“ stellen somit den Zusammenhang zwischen Echtzeit und Aufenthaltsort im Soundfile bzw. dessen Partialtonanalyse dar. Weiters gilt:

stretch_factor_A = A_stretch / (A_original – A_crossfade)

stretch_factor_B = B_stretch / B_original

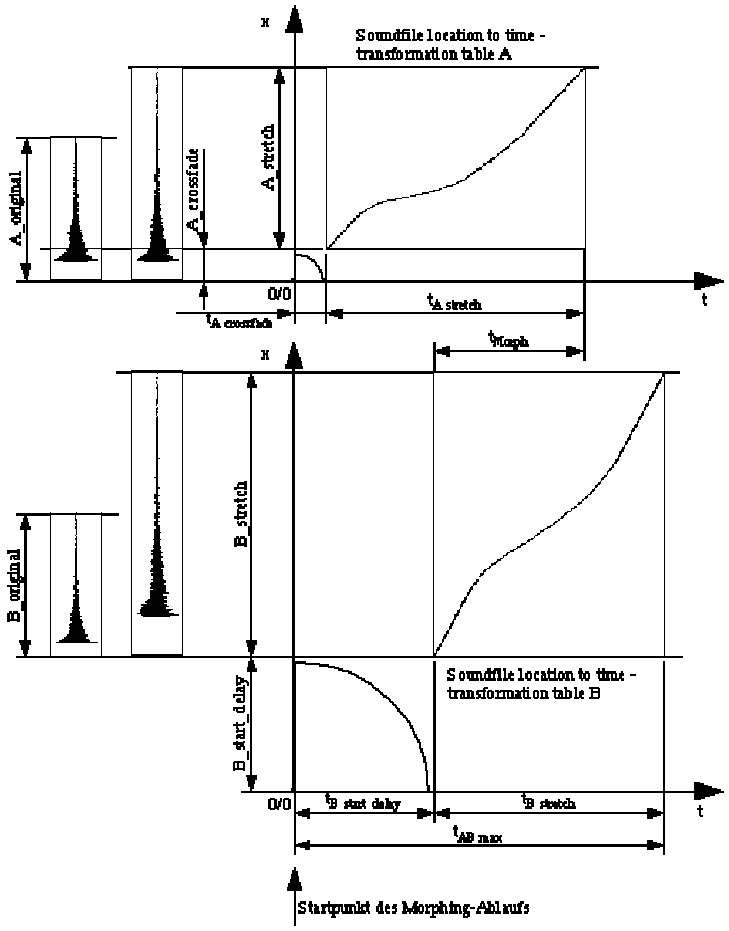

Abb. 8

In *Abb. 9* sieht man ein Oszillatorelement des Partialtons p einer ganzen Batterie von Oszillatoren der additiven Synthese zweier linearkombinierten Partialtonsätze. Am Eingang gelangen die Zeit/Ort-transformierten Werte der Echtzeit für die jeweilige additive Synthese von Klang A und B ein. Diese dienen zum Auslesen aus den Lookup-Tables von Amplitude und Frequenz des bestimmten Partialtons mit Index p. Die Frequenzen werden optional noch gefiltert (das Vibrato wird geglättet). Die Frequenz von Klang B wird noch transponiert um *transpose_B*, die Amplitude von Klang B wird noch verstärkt um *gain_B* (dient zur Anpassung beider Klänge). Die Frequenzpaare werden durch den aktuellen Wert von *freq_mor*ph (Frequenz A) und 1 minus *freq_morph* (Frequenz B) multipliziert und zusammengezählt. Die Amplitudenpaare werden durch den aktuellen Wert von *ampl_morph* (Amplitude A) und 1 minus *ampl_morph* (Amplitude B) multipliziert und zusammengezählt (Pendharkar [3]).

$$f_p = f_{pA} * freq\_morph + f_{pB} * (1 - freq\_morph)$$

$$a_p = a_{pA} * ampl\_morph + a_{pB} * (1 - ampl\_morph)$$

Die nun neu entstandenen Werte steuern Frequenz und Amplitude eines Oszillators. Die Signale aller Oszillatoren ergeben dann das Morphing-Signal der Klänge A und B.

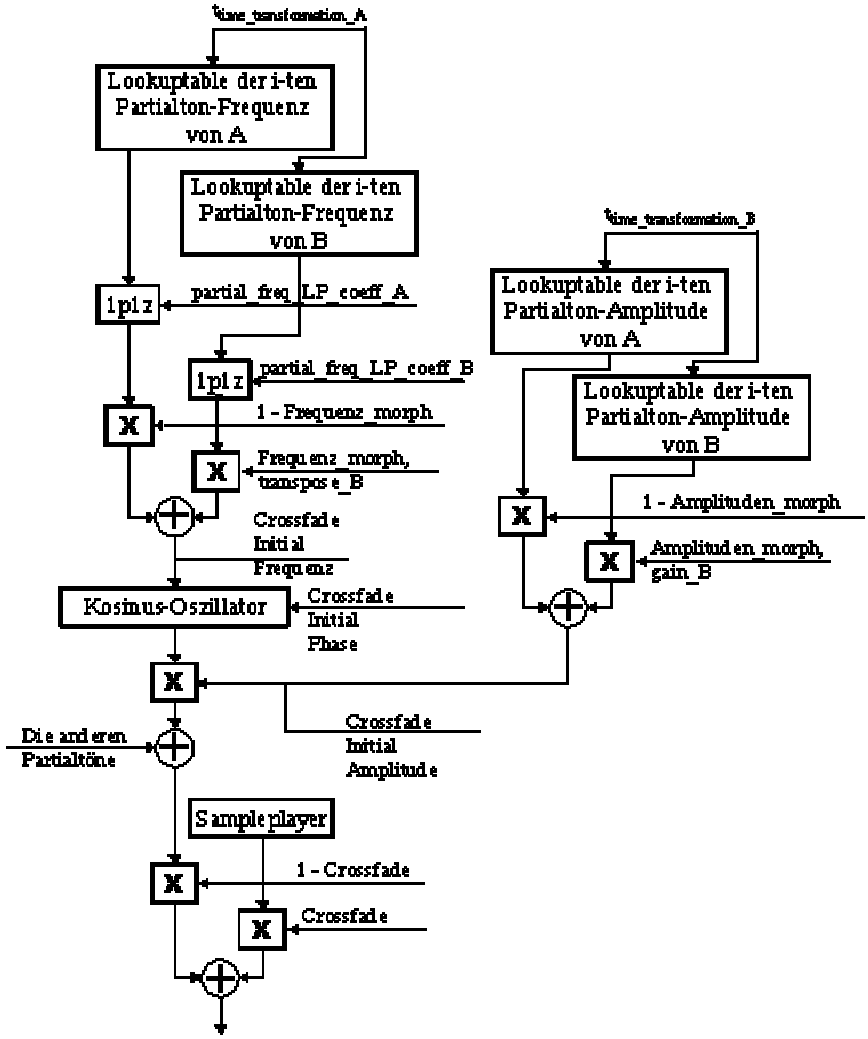

Abb. 9

Zum Zeitpunkt $t_{A\_crossfade}$ werden die Initialwerte von Amplitude, Frequenz und Phase direkt an die Oszillatoren geschickt, dann ihre zukünftigen Werte als Rampensteigung, und dann beginnt am Ausgang eine kurze Überblendung vom originalen Abspielsignal des Klangs A zur additiven Synthese des reinen Klangs A. Erst ab diesem Zeitpunkt darf man an Vermischungen mit Klang B und Glättungen von Frequenzen beginnen.

6 Implementierung der Morphing-Synthese in Pd

Der Pd-Patch *Partial_Synthesis_Morph.pd* basiert auf dem Modell der additiven Synthese von harmonischen Schwingungen. Diese Oszillatorenbank wird von zwei verschiedenen, miteinander verknüpfbaren Partialtonanalyse-Sätzen gesteuert. Die Voraussetzungen bezüglich Pd sind gleich wie beim vorherigen Analyse-Patch. Nach dem Starten von Pd-Patch *Partial_Synthesis_Morph.pd* (siehe *Abb. 10*) schaltet man 1.) den DSP-Toggle ein, 2.) initialisiert *number_of_oscillators*, 3.) initialisiert *samplerate* und *hoppsize*, 4.) editiert und versendet die Preset-Messagebox mit den wichtigsten Parametern (siehe *Abb. 11*), 5.) editiert die zeitlichen Dehn-/Stauch-Tables von A und B, die Frequenz- und die Amplituden-Morph-Table, 6.) filtert die Partialfrequenz-Tables von A, 7.) filtert die Partialfrequenz-Tables von B, 8.) startet den Morphing-Ablauf, .9.) speichert bei Zufriedenheit mit dem Ergebnis selbiges als Wave-Datei ab.

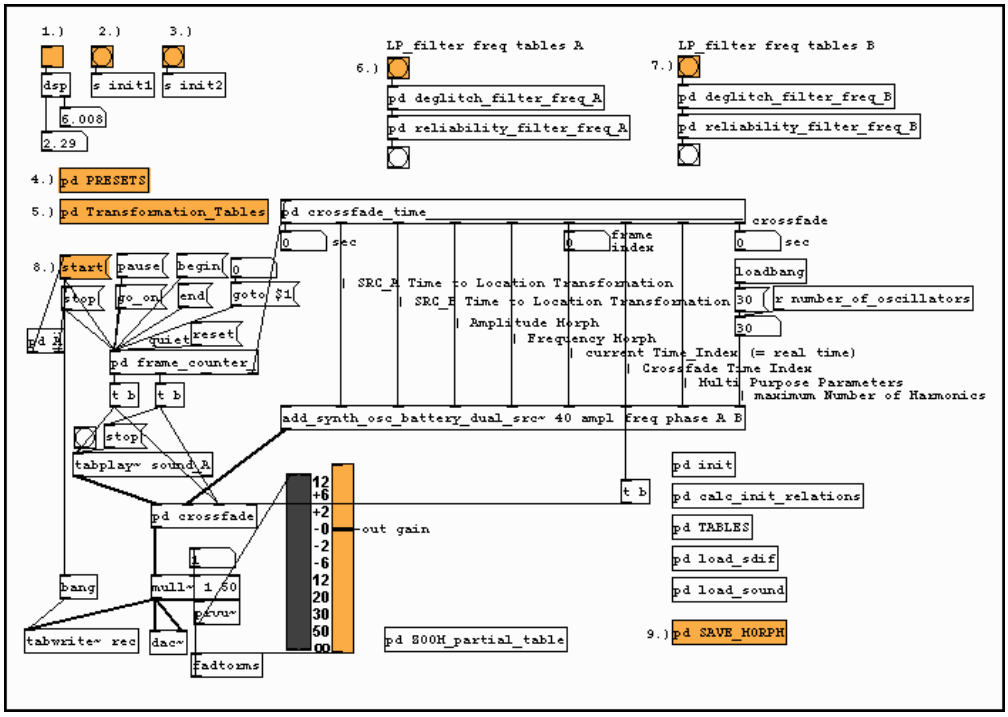

Abb. 10

ad 6.) und 7.):

Zuerst werden mittels eines Medianfilters diracförmige Störimpulse aus den Partialtonfrequenzen gefiltert, dass erst ab eines gewissen Verhältnisses zwischen Eingangswert zu Medianwert einsetzt. Dann werden die Partialtonfrequenzen bezüglich ihres Amplitudenwertes gefiltert, falls dieser unter einen Reliabilitätswert rutscht, schaltet sich ein Sample and Hold ein und überbrückt den unglaubwürdigen Wert durch seinen Vorgängerwert.

Ad „pd frame_counter“:

Dieser Zähler der Zeitquanten von *hoppsize* muss in einem eigenen Subpatch mit der Datenblockgröße von *hoppsize* sitzen, um dann alle relevanten Messages zum Zeitpunkt $t_{A\_Crossfade}$ richtig quantisiert zu versenden.

Die Parameter von *PRESETS* bedeuten:

```
;
number_of_partials_to_synthesize 80;
synthesis_crossfade_A 1.5;
center_B 1.5;
start_delay_B 2;
stretch_factor_A 1.5;
stretch_factor_B 2.4;
partial_freq_LP_coeff_A 0.01;
partial_freq_LP_coeff_B 0.01;
gain_B 0;
transpose_B -100;
reliability_threshold -70;
deglitch_bin_width 55;
src_sdif_name_A sounds/Klavier/es1.sdif;
src_wav_name_A sounds/Klavier/es1.wav;
src_sdif_name_B sounds/Sopran_a/a_l_e1.sdif;
sf_location_to_time_transform_A read
tables/transf_Klavier_es1.txt;
sf_location_to_time_transform_B read
tables/transf_Sopran_a_l_e1.txt;
freq_morph read
tables/freq_morph_Klavier_es1_Sopran_a_l_e1.txt;
ampl_morph read
tables/amp_morph_Klavier_es1_Sopran_a_l_e1.txt
```

Abb. 11

number of partials to synthesize: selbe Anzahl der Teiltöne wie bei der Analyse;

synthesis crossfade A: Zeitpunkt in sec. der Überblendung von Klangabspielung A zu additiver Synthese A = Beginn der Störimpulsfilterung in der Fundamentalfrequenz –Speichertabelle A;

center B: Beginn der Störimpulsfilterung in der Fundamentalfrequenz – Speichertabelle B;

start delay B: Verzögerungszeit des Startzeitpunkts von additiver Synthese B;

stretch factor A: Dehnungsfaktor der Abspielzeit von A;

stretch factor B: Dehnungsfaktor der Abspielzeit von B;

partial freq LP coeff A: Tiefpass-Koeffizient (0 .. 1) für Partialtonfrequenz-Glättungsfilter (Devibrato) von Klang A;

partial freq LP coeff B: Tiefpass-Koeffizient (0 .. 1) für Partialtonfrequenz-Glättungsfilter (Devibrato) von Klang B;

gain B: Korrekturpegel für Quelle B in dB;

transpose B: Korrekturtransposition für Quelle B in cents;

reliability_threshold: Zuverlässigkeits-Pegel in dB, der entscheidet,
ob Frequenzwerte mit logarithmischen Amplituden unter diesem Pegel
nicht mehr vertrauenswürdig sind, und somit durch den
glaubwürdigen Vorgängerwert ersetzt werden;

deglitch_bin_width: Filterbreite der Störimpulsfilterung der
Partialtonfrequenz-Tables in bins;

src_sdif_name_A: Quelldateiname der spektralen Teiltonverläufe A;

src_wav_name_A: Quelldateiname der Mono-Wave-Datei A;

src_sdif_name_B: Quelldateiname der spektralen Teiltonverläufe B;

sf_location_to_time_transform_A: ASCII-Quelldatei der
Ort/Zeit-Transformations-Table A
zum Dehnen/Stauchen des Zeitverlaufs von A.

sf_location_to_time_transform_B: ASCII-Quelldatei der
Ort/Zeit-Transformations-Table B
zum Dehnen/Stauchen des Zeitverlaufs von B.

freq_morph: ASCII-Quelldatei der Überblend-Table für Partialtonfrequenzenzen
zum Überblenden von Quelle A (unten) nach Quelle B (oben) über die Zeit.

ampl_morph: ASCII-Quelldatei der Überblend-Table für Partialtonamplituden
zum Überblenden von Quelle A (unten) nach Quelle B (oben) über die Zeit.

7 Pd-Bibliotheken

7.1 Objekte der External-Bibliothek *iem_morph*

read_sdif_info: berechnet die Länge einer in Partialtön zerlegten Datei;
Übergabeargumente: <symbol> Frame_Typ , in unserem Fall „HRM“;
Messages am ersten Eingang:
frame_typ Frame_Typ: in unserem Fall „HRM“;
read Dateiname: bewirkt die Ausgabe der Länge von Dateiname in sec.

read_sdif_to_tab: ladet die Daten der Partialtöne in eine Dreiergruppe von Tables (Partialamplitude, Partialfrequenz, Partialphase);
Übergabeargumente: <symbol> Frame_Typ, in unserem Fall „HRM“;

<symbol> Partialamplituden-Table-Namensrumpf;

<symbol> Partialfrequenzen-Table-Namensrumpf;

<symbol> Partialphasen-Table-Namensrumpf , alle 3 Rümpfe werden nach vorne hin mit einer Zahl (beginnend mit 1) und Unterstrich erweitert;

<float> maximale Anzahl von Harmonischen, soll mit der Analyse und Anzahl der Tables übereinstimmen;

Messages am ersten Eingang:

frame_typ Frame_Typ: in unserem Fall „HRM“;

read Dateiname: bewirkt die Übertragung der Informationen von der SDIF-Datei zu den Tables.

tab_amp_appendix Namensrumpf: stellt den zweiten Teil der Table-Namen für Partialamplituden-Tables dar;

tab_freq_appendix Namensrumpf: stellt den zweiten Teil der Table-Namen für Partialfrequenzen-Tables dar;

tab_phase_appendix Namensrumpf: stellt den zweiten Teil der Table-Namen für Partialphasen-Tables dar;

max_harm N_Partialtöne: bestimmt die maximal Anzahl an Teiltöne und damit die Anzahl der Tables mal 3;

n_frames N_Partialtöne: bestimmt die Mindestgröße der Tables;

sr_fftsize_hoppsize Samplerate + Fftsize + Hoppsize: informieren über Grundgrößen der Synthese;

write_tab_to_sdif: speichert die Daten einer Dreiergruppe von Tables (Partialamplitude, Partialfrequenz, Partialphase) im SDIF-Format ab;

Übergabeargumente: <symbol> Frame_Typ, in unserem Fall „HRM“;

<symbol> Partialamplituden-Table-Namensrumpf;

<symbol> Partialfrequenzen-Table-Namensrumpf;

<symbol> Partialphasen-Table-Namensrumpf , alle 3 Rümpfe werden nach vorne hin mit einer Zahl (beginnend mit 1) und Unterstrich erweitert;

<float> maximale Anzahl von Harmonischen, soll mit der Analyse und Anzahl der Tables übereinstimmen;

Messages am ersten Eingang:

frame_typ Frame_Typ: in unserem Fall „HRM“;

read Dateiname: bewirkt die Übertragung der Informationen von der SDIF-Datei zu den Tables.

tab_amp_appendix Namensrumpf: stellt den zweiten Teil der Table-Namen für Partialamplituden-Tables dar;

tab_freq_appendix Namensrumpf: stellt den zweiten Teil der Table-Namen für Partialfrequenzen-Tables dar;

tab_phase_appendix Namensrumpf: stellt den zweiten Teil der Table-Namen für Partialphasen-Tables dar;

max_harm N_Partialtöne: bestimmt die maximal Anzahl an Teiltöne und damit die Anzahl der Tables mal 3;

n_frames N_Partialtöne: bestimmt die Mindestgröße der Tables;

sr_fftsize_hoppsize Samplerate + Fftsize + Hoppsize: informieren über Grundgrößen der Synthese;

search_peaks: berechnet die relativen Maxima aus der Table *Betragsspektrum* und tragt sie in die Table „*Spitzenidizes*“ ein; ein relatives Maximum ist dann, wenn ein Wert größer gleich zu seinen linken und rechten Nachbar ist, und wenn er größer als der *Minimumpegel* ist, und wenn er größer als der Wert im „*geglättetes_Spektrum*“ plus „*additiver_Pegel*“ ist.

Übergabeargumente:

<symbol> Table-Name des Betragsspektrums;

<symbol> Table-Name des geglätteten Spektrums;

<symbol> Table-Name der Liste der Spitzenindizes;

Messages am ersten Eingang:

tab_mag Frame_Typ: Betragsspektrum

tab_floor Frame_Typ: geglättetes_Spektrum

tab_peak_freq Frame_Typ: Spitzenindizes

min_level Frame_Typ: Minimumpegel

add_level_floor_thresh “Additiver Pegel”:

<bang> : startet den Berechnungsvorgang und schickt die Anzahl der gefundenen Peaks an den Ausgang.

calc_accurate_peaks: berechnet die genauer interpolierten Werte für Amplitude, Frequenz und Phase der Maxima aus den Tables *Betragsspektrum,* Phasenrealteil und Phasenimaginärteil und tragt sie in die Table „Liste_der_*Spitzenidizes*“ ein; ein relatives Maximum ist dann, wenn ein Wert größer gleich zu seinem linken und rechten Nachbarn ist, und wenn er größer als der *Minimumpegel* ist, und wenn er größer als der Wert im „*geglättetes_Spektrum*“ plus „*additiver_Pegel*“ ist.

Übergabeargumente:

<symbol> Table-Name des Betragsspektrums;

<symbol> Table-Name des geglätteten Spektrums;

<symbol> Table-Name der Liste der Spitzenindizes;

Messages am ersten Eingang:

tab_mag Frame_Typ: Betragsspektrum

tab_floor Frame_Typ: geglättetes_Spektrum

tab_peak_freq Frame_Typ: Spitzenindizes

min_level Frame_Typ: Minimumpegel

add_level_floor_thresh “Additiver Pegel”:

<*bang*> : startet den Berechnungsvorgang und schickt die Anzahl *n_peaks* der

gefundenen Peaks an den Ausgang.

8 Literaturverzeichnis

[1] Miller Puckette, “*Pd Documentation Rel-0.39*“, Dokumentation von Pd im Netz: http://crca.ucsd.edu/~msp/Pd_documentation/

[2] A. M. Noll, “*Pitch determination of human speech by the harmonic product spectrum, the harmonic sum spectrum, and a maximum likelihood estimate*.“, Proc. Symp. Computer Processing in Comm. , pp. 779-798, 1969

[3] Chinmay Pendharkar, M. Gurevich, L. Wyse, “*Parameterized morphing as a mapping technique for sound synthesis*“, Proc. of the 9th Int. Conference on DAFx-06, Montreal, Canada, Sept. 2006