+ CD QCake Game Designer 0.5.8.1 • Pygame 1.7 • Scrolling Game Development Kit

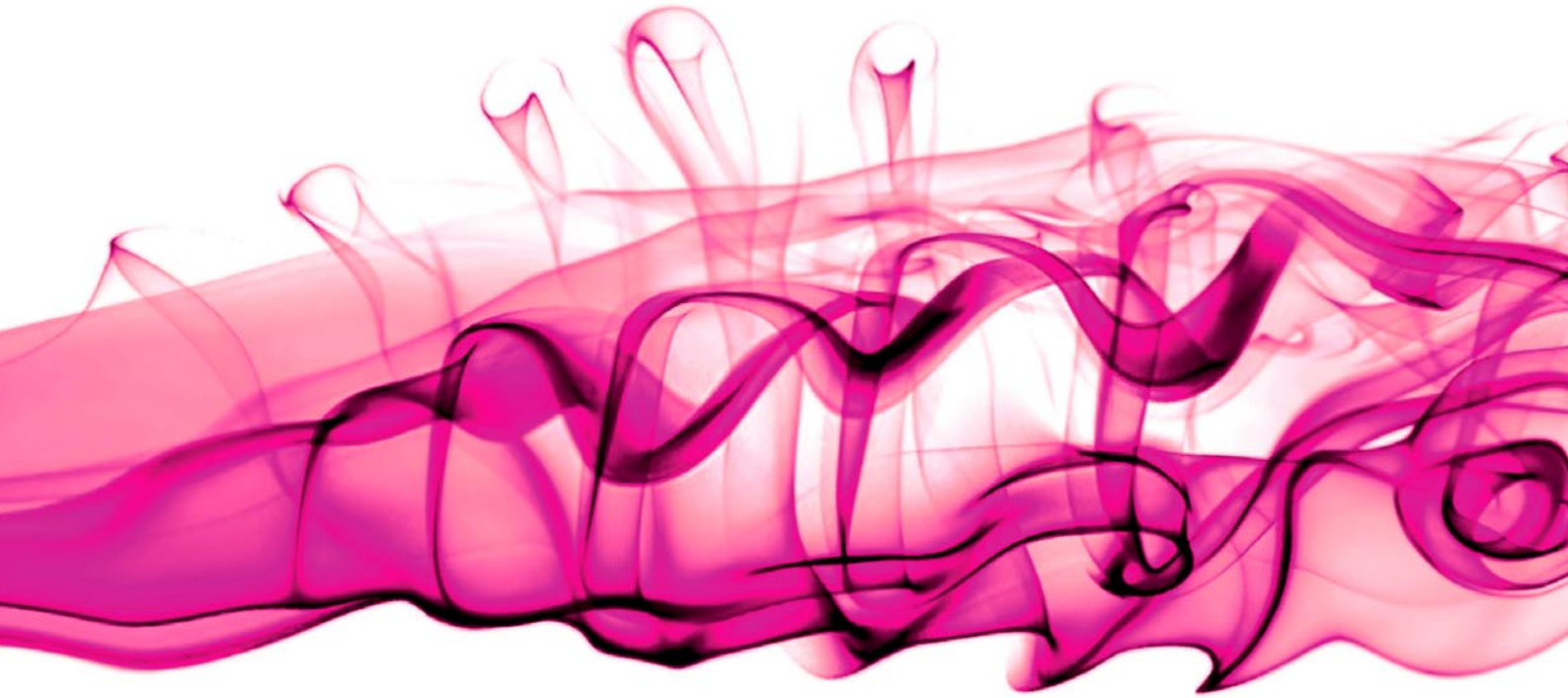
Nr. 4/2006 (10) Juni/Juli 2006 ISSN: 1734-7637 ZKZ 91700 Deutschland: 9,90 EUR Österreich: 11,40 EUR Luxemburg: 11,40 EUR Schweiz: 19,40 CHF

SDJ EXTRA

Programmieren von Spielen und Grafiken

Audio-Zeit- und -Pitch-Skalierung

Olli Parviainen



Audio-Zeit- und -Pitch-Skalierung

Olli Parviainen

Jeder, der schon einmal Sounds aufgenommen hat, kennt den Effekt, der entsteht, wenn man die Aufnahme mit doppelter Originalgeschwindigkeit abspielt. Eine Aufnahme so abzuspielen halbiert zwar die Spieldauer, bewirkt aber gleichzeitig einen recht offentichtlichen Nebeneffekt, bei dem der Sound eine Oktave nach oben gepitcht wird; was menschliche Stimmen zu lustig klingenden Stimmen, wie von Cartoon-Figuren werden lässt. Folgerichtig wir die Aufnahme bei einer langsameren Abspielgeschwindigkeit länger dauernd und der Sound wird nach unten gepitcht sein. In den Tagen der analogen Audio-Technologien, als Band-Rekorder und Vinyl-Plattenspieler benutzt wurden, konnte man diesen Effekt sehr leicht erreichen, indem man einfach falsche Geschwindigkeitseinstellungen für das Abspielen einstellte. In der digitalen Signalverarbeitung erzeugt man den Effekt normalerweise über Resampling als Zwischenschritt. Dieses Resampling beeinflusst aber beide Größen, Zeit und Pitch, in gleicher Weise. Eigentlich wäre es viel angenehmer, wenn man Sound-Dauer und -Pitch getrennt voneinander kontrollieren könnte, so dass Zeitachse modifiziert werden könnte, ohne den oftmals höchst unerfreulichen Effekt der cartoonhaften Stimmen durch das Resampling zu erhalten. Glücklicherweise existieren Techniken in der Signalverarbeitung, die es ermöglichen, Zeit- und Pitch-Skalen unabhängig voneinander zu vearbeiten – diese Techniken werden als Zeit/Pitch-Skalierung, Zeit/Pitch-Verschiebung oder Zeitdehnung bezeichnet.

Anwendungen

Die Zeit/Pitch-Skalierung bietet jede Menge Anwendungen. Indem man Zeitskalierung benutzt, um die Geschwindigkeit von Musik zu verlangsamen, kann man sich das Üben mit Musikinstrumenten vereinfachen oder leichter Tanzschritte einstudieren. Aufgenommene Sprache zu verlangsamen kann das Niederschreiben von gesprochenen Notizen erheblich vereinfachen und blinde Menschen, die sich ein Audio-Buch anhören könnten die Sprache schneller abspielen wollen, um Zeit zu sparen. Weiterhin muss man beim Konvertieren zwischen Videoformaten mit unterschiedlicher Framerate die Feinjustierung bedenken. Wenn beispielsweise ein Kino-Format 24 Frames/Sekunde in ein TV-Format mit 25 Frames/Sekunde umgewandelt wird, wird das die Bild-Spielzeit des Films um 4% verringern; hier muss die Tonspur natürlich entsprechend angepasst werden. Ähnlich kann die Änderung des Sound-Pitches eine angenehmere Tonlage ergeben, die für Karaoke oder Gesangsübungen benutzt werden, um sich der Tonlage des Sängers anzunähern. Wenn man mit einem Instrument zu laufender Musik spielt, kann es oft einfacher sein, die Tonart der Hintergrundmusik zu ändern, als jedesmal das Instrument umzustimmen. Pitch-Korrektur ist heutzutage auch eine Standardmethode beim Aufnahmeverfahren im Studio um falsche Töne im Gesang zu korrigieren; so kann man einfach etwaige Lücken im gesanglichen Talent ausgleichen, um auf dem Musikmarkt bestehen zu können. Zuletzt sollen auch Menschen nicht unerwähnt bleiben, die, für ihre zweifelhaften Pläne, mit Hilfe einer Änderung des Pitches eine Veränderung der Stimme bewirken wollen, um ihre wahre Identität zu verbergen.

Ansätze

Bei der allgemeinen Betrachtung der Time/Pitch-Skalierungs-Methoden, die mit jeder Art von abgetastetem Sound funktionieren, ob es nun Sprache, Musik oder anderer Sound ist, gibt es zwei Ansätze um solche Effekte zu implementieren; nämlich die Abwicklung über den Zeit-Wert oder über den Frequenz-Wert. Die Zeitwert-Methode handelt direkt mit den abgetasteten Daten, beispielsweise dem SOLA-Algorithmus, der in diesem Artikel vorgestellt wird. Der Vorteil dieser Vorgehensweise ist vor allem, dass die Einbindung sehr direkt geschieht; die Sound-Daten werden im gleichen Format manipuliert, in dem sie auch abgetastet wurde und in dem sie abgespielt wird. Ein Nachteil dieser Methode besteht darin, dass sie sehr dazu neigt widerhallende Artefakte zu produzieren, die deutlicher werden, wenn die Veränderung durch die Modifikation des Originalsounds 15% oder mehr ist. Der andere Ansatz ist die Benutzung einer Methode, die den Frequenzwert modifiziert. So ist es möglich abgetasteten Sound in kurzzeitige Frequenz/Amplituden-Komponenten und mit Frequenzinformationen zu skalieren. Phase Vocoder (dt. *Sprachentschlüsselungsgerät*), der weiter unten besprochen wird, ist ein Beispiel für diese Methode. Der Vorteil dieser Methode ist, dass sie es erlaubt, viel raffiniertere und anspruchsvollere Mo-

Olli Parviainen ist Embedded-Software-Designer, Projektmanager und Erfinder der Open-Source Audio-Time/Pitch skalierenden Bilbiothek SoundTouch.
Kontakt: *olli.parviainen@iki.fi*

Fast Fourier Transformation (FFT)

Fourier Transformation ist ein Algorithmus, der ein Array von N Teilen zeitlich begrenzter Abtastraten in Frequenzbereiche umwandelt, daraus resultieren Signalamplitude und Phaseninformation für N/2 äquidistante Frequenzbereich-Kästen, jedes der Größe [Abtastfrequenz] / (N * 2) Hz. Die Fourier Transformation wird gewöhnlich in Applikationen verwendet, die sich auf Frequenzbereich-Verarbeitung verlassen oder mit dem Signalleistungsspektrum arbeiten. Während die Berechnung von Fourier Transformation bei großen Array-Größen N schwierig wird, existiert eine schnellere Variante der Transformation, Fast Fourier Transformation (FFT), welches die Menge der Kalkulationen radikal reduzieren kann, unter der Vorraussetzung, dass die Datengröße N eine Integer-Potenz von 2 sein muss, das bedeutet N = 2^i, d.h. N = 2, 4, 8, 16.. usw. Praktische Anwendungen verwenden fast immer die FFT-Variante dank ihres Berechnungsvorteils. FFT hat auch eine umgekehrte Funktion, inverses FFT oder IFFT, welches die Frequenzdomänen-Information in das zeitbegrenzte Abtastformat umwandelt.

digfikationen am Sound durchzuführen, die stärker der eigentlichen Hörerfahrung des Menschen ähneln, da das menschliche Gehör ebenfalls mit Frequenz/Amplituden arbeitet; wir haben keine Magnetspule oder einen digitalen Sampling-Schaltkreis in unseren Ohren. Unser Sinnesapparat Ohr besteht jedoch aus vielen, winzigen Sensoren, von denen jeder für einen gewissen Frequenzbereich zuständig ist; zusammen kreieren sie ein großartiges Hören durch alle Frequenzbereiche (zumindest so lang, bis einst perfekte Ohren durch sehr laute Hobbies zerstört werden: Rockmusik, Motorsport, Kindererziehung). Wie auch immer... trotz ihrem Potential und ihrer Eleganz ist die Frequenzwertmethode ungleich komplexer zu programmieren und zu errechnen als Zeitwertmethoden, was sie unpraktikabel macht, wenn eine Anwendung von Rechenleistung abhängig ist.

SOLA

SOLA, die Abkürzung für Synchronous-OverLap-Add, arbeitet, indem es die Sounddaten in kleine Sequenzen von zwischen zehn bis 100 Millisekunden zerschneidet, und diese dann wieder zusammenfügt; entweder mit einem angemessenen Teil von Daten entfernt, oder durch Wiederholung hinzugefügt, um eine kürzere bzw. längere Abspielzeit zu erhalten als im Original vorhanden. Algorithmen, die hierauf basieren werden gelegentlich auch als TDHS (*Time-Domain Harmonic Sampling*), WSOLA oder PSOLA bezeichnet, je nachdem, wie der die Überlappung genau stattfinden soll, wie unten besprochen. Um zu auffällige Störungen im Sound an Stellen zu verhindern, an denen zwei benachbarte Sequenzen zusammengefügt werden, werden die Sequenzen partiell überlappt, während die Sound-Amplitude in der Überlappung schrittweise von einer Sequenz zur nächsten geschoben wird. Deswegen heißt es „Overlap-Add“ in SOLA. Als einfachste Möglichkeit könnte die SOLA-Implementierung eine gewöhnliche Sequenzlänge benutzen und die Sequenzen in gleichen Intervallen aus der Originaldatei nehmen. Die Soundlänge soll 10% kürzer sein? Ok, wir benutzen eine Prozess-Sequenzdauer von beispielsweise 100 Millisekunden (+ Überlappungszeitrahmen), nehmen die Sequenzen aus der Original-Sounddatei mit 110 Millisekunden, fügen diese Sequenzen durch Überlappung zusammen und schon haben wir, was wir wollten. Auf ähnlichem Wege können wir eine um 10 % erhöhte Dauer erreichen, indem wir 100 Millisekunden lange Sequenzen der Originaldatei zu Intervallen von 90 Millisekunden nehmen, und so weiter. Wie auch immer: Die praktische Implementierung von SOLA ist nicht ganz so einfach. Die Sequenzen auszusuche ohne zu beachten, welchen Inhalt sie haben, kann leicht zu Sequenzen führen, die nicht zu gut genug zu einander passen und so eine schreckliche Tonqualität mit Brummen und Klickgeräuschen erzeugen. Die rührt daher, dass die Unterschiede im Ablauf zu groß sind, auch nach Anwendung der Überlappung.

Überlegungen zur Implementierung

Um eine zufriedenstellende Sound-Qualität zu erreichen müssen bei der Implementierung von SOLA die verarbeitenden Sequenzen so gefählt werden, dass die Wellenformen der angrenzenden, zusammenzufügenden Sequenzen in der Überlappungszeit so gut wie irgend möglich zueinander passen. In der Praxis wird der Soundstrom pro Sequenz verarbeitet, wobei die nächste verarbeitete Sequenz aus einer Auswahl von möglichen ausgleichenden Sequenzen gewählt wird, damit die beiden Sequenzen bestmöglich aufeinander abgestimmt sind. Eine gute Möglichkeit um diese bestmöglich zueinander passenden Sequenzen auszuwählen ist, eine Funktion zur Wechselbeziehung zwischen dem Ende der vorhergehenden Sequenz und der Nachfolgekandidaten zu berechnen; die Gegenrechnung bei der die beiden Wellenformen sich am meisten ähneln gibt dann den höchsten Wechselbeziehungswert zurück. Die Sequenzen werden dann durch Überlappung zusammengefügt, um einen neuen Soundstream zu erzeugen, der eine andere Dauer als das Original hat. Der ganze SOLA-Algorithmus ist in Abbildung 1 gezeigt. Bedenken Sie, dass die Zeitachsen in diesem Bild beliebig sind und nicht unbedingt wirklichen Zeiteinheiten entsprechen müssen. Während des Ausführens des Algorithmus wird der Originalsound in verarbeitende Sequenzen von passender Dauer zerlegt; neue Sequenzen werden in entsprechenden Abständen nach der vorhergehenden Sequenz gewählt, um das im Endeffekt gewünschte Zeitverhältnis zu erreichen. In Bild 1 beginnt die erste Verarbeitungssequenz am Zeitpunkt 0, während die Dauer sieben Zeiteinheiten beträgt, inklusive der beiden Überlappungsräume an Beginn und Ende von jewils zwei Einheiten.

Jede weitere Sequenz wird in Abständen von nominell neun Einheiten gewählt, was in diesem Fall zu einem Zeitskalierungsverhältnis von (7-2) / 9 = 0.555, also ~ 44.5% Reduzierung der Zeitdauer im Vergleich zum Originalsound führt. Statt aber die neue Sequenz direkt nach dem nominell kalkulierten Zeitabstand zu nehmen, wird der Beginn tatächliche innerhalb eines Zeitfensters um den Wert herum gewählt, in unserem Beispiel zwischen den Werten 8 bis 10, damit die beiden Sequenzen, sind die durch Überlappung verbunden, so gut wir möglich zusammenpassen. So entsteht aus den beiden ein fließender Klang. In Bild 1 kann kann man die resultierende Zeitkompression am Ver-

gleich der originalen und der entstandenen Waveform abgelesen werden. Die Wechselbeziehungsfunktion funktioniert für die Erarbeitung von Ähnlichkeiten zwischen den Sequenzwelleformen sehr gut. Es werden aber auch andere Methoden zur Ähnlichkeitsmessung vorgeschlagen. Eine davon ist, die Enden der Sequenz über den Beat anzupassen, was den Wiederhall verhindern kann, vorrausgesetzt der Beat kann durchgängig gut herausgefiltert werden. Eine weitere Alternative zum Finden der besten Überlappung wäre das Betrachten der Spektralähnlichkeiten anstatt deren der Wellenformen.

Multi-Kanal-Sound

Wenn man mit Stereo-, Surround- oder anderen Multi-Kanal-Sounds arbeitet, tut man gut daran, alle Kanäle am gleichen Zeitpunkt überlappen zu lassen. Auch wenn es im ersten Moment einleichtend klingt, n-Kanal Sound einfach als n verschiedene Mono-kanäle zu betrachten, wäre das kein brauchbarer Ansatz, denn die verschiedenen Kanäle würden an leicht unterschiedlichen Zeitpunkten überlappt werden, je nach spezieller Beschaffenheit des einzelnen Kanals, was dazu führen würde, dass die Kanäle ihre fein abgestimmte Synchronität verlieren, die die grundlegende Basis von Stereo- oder Surround-Sound bildet. Statt dessen muss die Auswahl der Überlappungszeitpunkte als Kompromiss aller betroffenen Kanäle gesehen und so ausgeführt werden. Das kann erreicht werden, indem man nach Summierung aller Kanäle, mit Hilfe der oben besprochenen Wechselbeziehungsfunktion die erhaltenen Daten auswertet und den erhaltenen Wert auf alle Kanäle anwendet. Die SOLA-Implementierung ermöglicht die grundlegenden Zusammenzählen/Multiplizieren Operationen der Arithmetik; das bedeutet, dass SOLA entweder mit Integer- oder Fließkomma-Arithmetik benutzt werden kann.

Pitch-Skalierung mit SOLA

SOLA als solches erzeugt einen Zeitskalierungseffekt, wie er oben beschrieben wurde. Allerdings kann die Kombination des SOLA-Algorithmus mit einem Resampling-Effekt, der die Zeit- und Pitch-Achse im gleichen Verhältnis modifiziert, mit einem kleinen Mehraufwand, auch einen Pitch-Verschiebungs-Effekt produzieren. Dies funktioniert folgendermaßen:

- Benutzen Sie das Resampling, um den Pitch mit dem gewünschten Wert zu verringern oder zu erhöhen. Da Resampling Dauer und Pitch eines Sounds gleichmaßen verändert, wird die Sounddauer im Vergleich zum Original länger werden.
- Dann benutzen Sie den SOLA-Algorithmus um die Sounddauer, die durch das Resampling verändert wurde, wieder zu ihrem Originalwert zurückzuändern. Das Ergebnis hat zwar die gleich Dauer, wie das Original, aber einen anderen Pitch.

Es können beliebige Resampling + SOLA Zeitskalierungsraten angewendet werden, um beliebige Pitch-Skalierungsraten oder sogar einen beliebigen simultanen Zeit- und Pitchskalierungseffekt zu erlangen, wenn es erwünscht ist. Um beispielsweise den Pitch mit einem Verhätnis von zwei (also einer Oktave) zu erhöhen, muss man die Sounddauer zuerst durch Resampling auf die Hälfte des Originals bringen, was uns einen doppelt so hohen Pitch bringt, bevor man mit SOLA die Sounddauer mit dem gleichen Faktor (zwei) verlängern läst, was zu dem Originalsound führt, der jetzt eine Oktave höher gepitcht ist. Auf ähnliche Weise, können wir einen niedrigeren Pitch erreichen, indem wir erst für längere Dauer und niedrigeren Pitch resamplen und dann mit SOLA den entsprechenden Wert benutzen, um die Dauer wieder dem Original anzugleichen. Schon haben wir einen Sound mit niedrigerem Pitch, aber gleicher Länge.

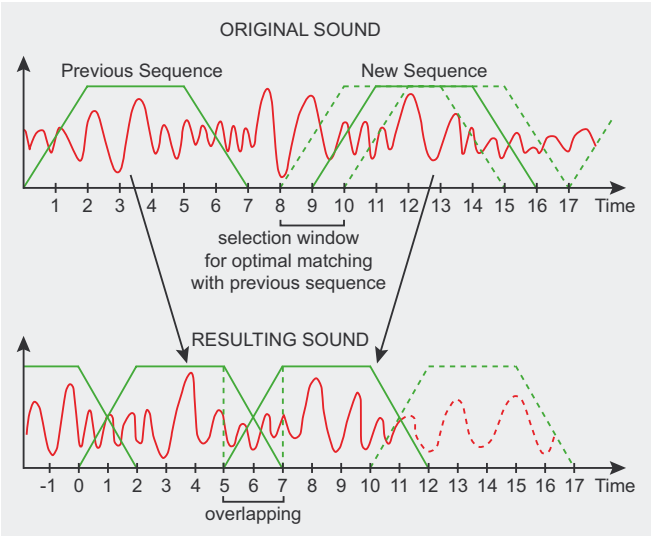


Abbildung 1. *SOLA Algorithmus*

Sound-Artefakte und Parameter

Ein typisches Sound-Artefakt, das von SOLA produziert werden kann ist ein Echo oder widerhallender Klang. Das ist vielleicht bei kleineren Änderung wenig signifikant, jedoch wird es deutlicher, wenn man größere Zeiten verlangsamen will. Um dieses zu verringern, ist die Dauer der verarbeitenden Sequenz ideal so, dass sie relativ nah an der grundlegenden Frequenz im Sound anliegt – zum Beispiel könnte eine betimmte Frequenzkomponente oder ein konstanter Beat als Referenz für die Auswahl der idealen Sequenzdauer sein. Auch die Breite des Auswahlfensters für die ausgleichenden Sequenzen kann die Soundqualität beeinflussen. Je breiter dieses Fenster, desto höher ist die Wahrscheinlichkeit, dort einen passenden Teil für die überlappenden Sequenzen zu finden. Ist das Fenster jedoch zu groß, wird sich der erhaltene Sound instabil anhören, als würde er herumrutschen. Zum Überlappen reicht meist ein Bruchteil der ganzen Länge der Sequenz aus.

Normalerweise arbeitet lineares Amplitudengleiten über die Überlappungsperiode wie die Fensterfunktion. Die Ablaufslatenz des SOLA Algorithmus hängt von der Dauer der Verarbeitungssequenz, der Größe des Fensters für die suchende Überlappung und der Größe der Überlappungsperiode ab, welche sich normalerweise auf bis zu 100 Millisekunden addiert. Diese Ablauflatenz ist normalerweise kein Problem, wenn man Sounddateien offline ausführt, aber wenn die Applikation strikte Anforderungen zur Soundsynchronisation hat oder wenn man Sound in Echtzeit aus-

Listing 1. *SOLA example routine*

```
// Zeitskalierungsfaktor, Werte > 1.0 erhöhen, Werte
// < 1.0 verringern Tempo
#define TIME_SCALE      0.85    // 15% slower tempo
// Verarbeitungssequenzgröße (100 msek mit 44100Hz
// Abtastrate)
#define SEQUENCE        4410
// Überlappungsgröße (20 msek)
#define OVERLAP         882
// Bester Überlappungspunkt Suchfenster (15 msek)
#define SEEK_WINDOW     662
// Verarbeitungssquenz flacher Mittelabschnittsdauer
#define FLAT_DURATION   (SEQUENCE - 2 * (OVERLAP))
// Theoretisches Intervall zwischen den
// Verarbeitunsseqzuenzen
#define SEQUENCE_SKIP   ((int)((SEQUENCE - OVERLAP)
      * (TIME_SCALE)))
// Abtasttyp, 16bit Vorzeichen Integer
typedef short SAMPLE;
// Benutze die Kreuzverbindungsfunktion um den besten
// Überlappungspunkt zu finden wo input_prev und input_nw am
// besten zusammen passen
int seek_best_overlap(const SAMPLE *input_prev,
      const SAMPLE *input_new) {
   int i;
   int bestoffset = 0;
   // willkürlich großer negativer Wert
   float bestcorr = -1e30f;
   float temp[OVERLAP];
   // Vorberechnung überlappender Neigungen mit input_prev
   for (i = 0; i < OVERLAP; i ++) {
      temp[i] = (float)(input_prev[i] * i * (OVERLAP - i));
   }
   // Finde den besten Überlappungspunkt innerhalb
   // [0..SEEK_WINDOW]
   for (i = 0; i < SEEK_WINDOW; i ++) {
      int j;
      float crosscorr = 0;
      for (j = 0; j < OVERLAP; j ++) {
         crosscorr += (float)input_new[i + j] * temp[j];
      }
      if (crosscorr > bestcorr) {
         // Neuen besten Punkt-Kandidaten gefunden
         bestcorr = crosscorr;
         bestoffset = i;
      }
   }
   return bestoffset;
}
// Überlappe 'input_prev' mit 'input_new' durch Verschieben
// der Amplituden während Überlappungsabtastungen.
// Speichere Ergebnis in 'output'.
void overlap(SAMPLE *output, const SAMPLE *input_prev,
         const SAMPLE *input_new) {
   int i;
   for (i = 0; i < OVERLAP; i ++) {
      output[i] = (input_prev[i] * (OVERLAP - i)
            + input_new[i] * i) / OVERLAP;
   }
}
// SOLA-Algorithmus. Führt Zeitskalierung für Abtastdaten
// in 'input' durch, schreibt das Ergebnis in 'output'.
// Gibt die Zahl der output-Abtastungen zurück.
int sola(SAMPLE *output, const SAMPLE *input,
       int num_in_samples) {
   int num_out_samples = 0;
   const SAMPLE *seq_offset = input;
   const SAMPLE *prev_offset;
   while (num_in_samples > SEQUENCE_SKIP + SEEK_WINDOW) {
      // Kopiere flache Mittelsequenzen von der aktuell
      // verarbeiteten Sequenz in output
      memcpy(output, seq_offset, FLAT_DURATION
            * sizeof(SAMPLE));
      // Berechne einen Zeiger zum Überlappen am Ende der
      // verarbeiteten Sequenz
      prev_offset = seq_offset + FLAT_DURATION;
      // Aktualisiere den Input-Zeiger zum theoretischen
      // Beginn der nächsten Verarbeitungssequenz
      input += SEQUENCE_SKIP - OVERLAP;
      // Suche den wirklich am besten passenden Punkt
      // unter Verwendung von Kreuzverbindung
      seq_offset = input + seek_best_overlap(
            prev_offset, input);
      // Überlappe zwischen vorheriger und neuer Sequenz,
      // Kopiere das Ergebnis in output
      overlap(output + FLAT_DURATION, prev_offset,
            seq_offset);
      // Aktualisiere input & Sequenzzeiger mit dem
      // Überlappungswert
      seq_offset += OVERLAP;
      input  += OVERLAP;
      output += SEQUENCE - OVERLAP;
      num_out_samples += SEQUENCE - OVERLAP;
      num_in_samples -= SEQUENCE_SKIP;
   }
   return num_out_samples;
}
```

führt oder abspielt, kann es nötig sein, der Latenz Aufmerksamkeit zu schenken. Ein Beispiel für solche latenzkritischen Applikationen ist eine Echtzeitanwendung, die Sound mit einem Mikrofon aufnimmt, dann den aufgenommenen Sound in Echtzeit mit SOLA bearbeitet, um einen Pitch-Verschiebungseffekt zu erzeugen und dann sofort den bearbeiteten Sound auf einem Lautsprecher abspielt. In einem solchen Fall ist es gut zu bemerken, dass der SOLA Bearbeitungsschritt eine zusätzliche 100 Millisekunden-Verzögerung zwischen den Aufnahme- und Wiedergabestufen einfügt.

SOLA Beispielquelltext

Listing 1 zeigt eine einfache Implementation einer SOLA-Routine. Die Funktion *sola()* bearbeitet abgetastete Sounddaten, die im Array 'input' übergeben werden und speichert

den daraus resultierenden zeitskalierten Sound im 'Output'-Array, davon ausgehend, dass beide Arrays für die gewünschte Verarbeitung groß genug sind. Der Algorithmus geht davon aus, dass die Sounddaten Monosound sind, der bei einer Rate von 44100 Hz mit 16 bit Integer-Sample-Format abgetastet wird. Die Zeitskalierungsrate ist in #define TIME_SCALE angegeben, der Standardwert von 0,85, entsprechend dazu (1.0 - 0.85) * 100% = 15% längere Dauer als original. Benutzer können diesen Wert verändern, um andere Zeitskalierungsraten zu erreichen. Die Subroutine *overlap()* überlappt die zwei angegebenen Eingabesequenzen, indem sie über den Überlappungszeitraum schrittweise das Amplitudengewicht von einer Sequenz zur anderen schiebt. Die Subroutine *seek_best_overlap()* sucht nach dem besten Überlappungszeitpunkt, indem sie Kreuzverbindungen (engl. *cross-correlation*) zwischen den Soundsequenzen *input_prev* und *input_new* während ihrer Überlappungszeiträume herstellt, indem sie verschiedene Verschiebungen für *input_new* innerhalb der Spanne [0..SEEK_WINDOW] testet. Ein Vorkalkulationsschritt, der die *input_prev*-Überlappungsregion mit den Überlappung-Verschiebungskoeffizienten multipliziert, wird vor der eigentlichen Kreuzverbindung durchgeführt, da eine solche Vorkalkulation die eigentliche Kreuzverbindungsberechnung deutlich beschleunigt. Beachten Sie, dass Kreuzverbindung-Ergebnisskalierung durch die Vektorlängen unterlassen wird, da es in diesem Fall ausreichend ist, den größten Verbindungswert zu finden, ohne um sich die tatsächliche Ergebnisgröße zu kümmern. Kreuzverbindungsberechnung benutzt der Einfachheit halber Fließkommaarithmetik; obwohl die Kreuzverbindungsberechnung auch reine Integer-Arithmetik verwenden könnte, wären in einem solchen Fall zusätzliche Subergebnis-Skalierungen nötig, um sicherzustellen, dass Integer-Overflows nicht auftreten können.

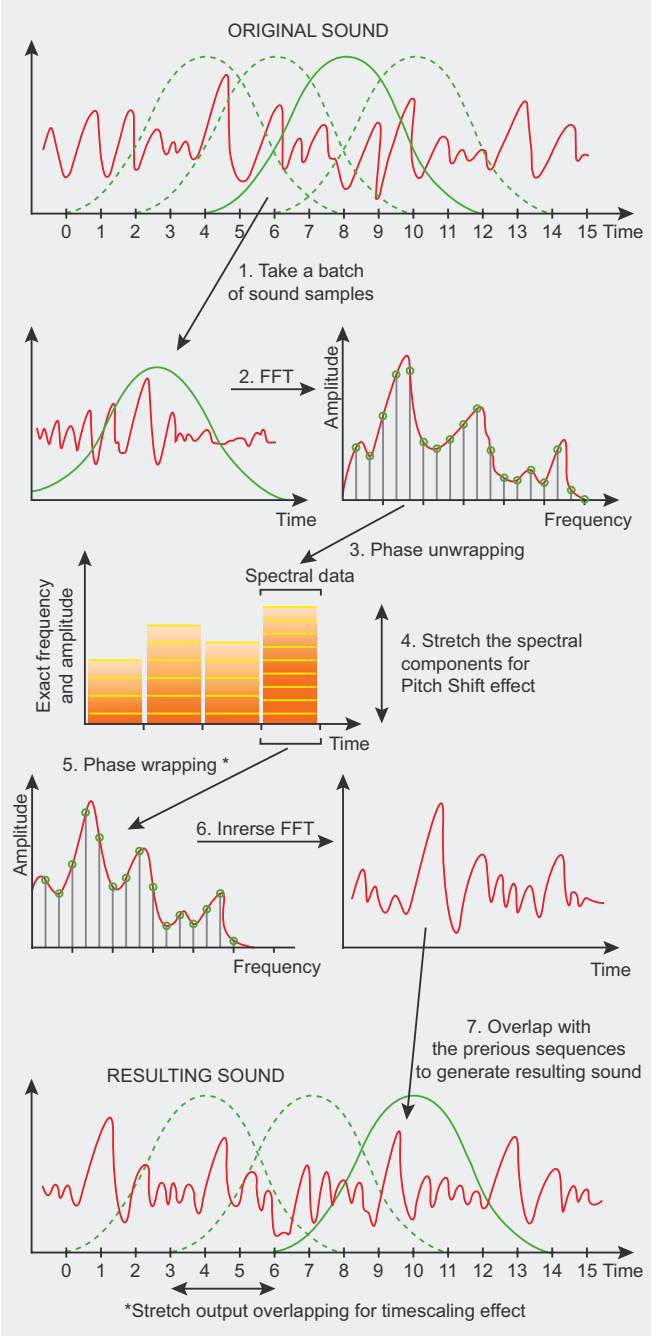


Abbildung 2. *Phase Vocoder Algorithmus*

Phase Vocoder

Phase Vocoder ist ein Algorithmus, der zeitlich begrenzt abgetasteten Sound in Frequenz/Amplituden-Präsentation und zurück umwandelt, dabei ermöglicht, die Soundcharakteristiken in der Frequenzdomäne zu bearbeiten.

Für einen Überblick über den Phase Vocoder Algorithmus werfen Sie einen Blick auf Abbildung 2. Zeit/Pitch-Skalierung unter Benutzung von Phase Vocoder besteht aus drei primären Verarbeitungsstufen:

- Analyse – wandelt Sound in Frequenz/Amplituden-Präsentation um,
- Effekt – wendet Effekte auf die Frequenz/Amplituden-Daten an,
- Synthese – konvertiert Soundinformationen vom Frequenz/Amplituden-Format zurück zum Abtast-Format.

Analyse und Synthese

In der Praxis sind die Phase Vocoder Analyse- und Synthesestufen unter Verwendung des Fast Fourier Transform (FFT) Algorithmus implementiert, der ein gut etablierter und effizienter Algorithmus zur Konvertierung zwischen Zeit- und Frequenzdomänen ist. Sounddaten werden in Sequenzen verarbeitet, indem eine passende Menge Samples vom ursprünglichen Sounddatenstrom genommen und eine Fensterfunktion und FFT auf diese Daten angewandt wird. Sehen Sie sich die Stufen 1 und 2 in Abbildung 2 an. Allerdings transformiert FFT zeitlich begrenzt abgetastete Daten in komplex-gewertete (engl. *comlex-valued*) Phase+Amplitude-Komponenten mit äquidistanten Frequenzkästen (äquidistant = gleich weit voneinander entfernt). Ach, diese äquidistanten Frequenzkästen mit komplex-gewerteten Inhalten sind kein passendes Format zur Modifizierung von Zeit/Pitch Skalacharakteristiken, sondern ein zusätzlicher Analyseschritt, Phasenauspacken (engl. *phase unwrapping*) genannt, wird benötigt, um die komplexgewerteten Phase+Amplitude-Informationen von vordefinierten Frequenzen weiter in exakte Frequenz+Amplituden-Komponenten zu konvertieren. Wenn man zum Beispiel ein FFT der Größe 1024 auf Sound mit der Abtastrate von 44100 Hz anwendet, erlangt der FFT komplex-gewerte-

te Phase/Amplitude-Komponenten für gleichmäßig verteilte Frequenzkästen, jede mit einer Größe von 44100/(2*1024) ~ 21.5 Hz. Dies wäre dann die primäre Frequenzauflösung der FFT'en Daten, was bedeutet, dass, wenn es eine Frequenzkomponente von sagen wir 30 Hz in den Audiodaten vorhanden gäbe, würde dies weder genau zum 21,5 Hz Kästen noch zum benachbarten Kästchen von 43,0 Hz ausgerichtet sein, sondern die 30 Hz Signalleistung würde zwischen diesen 21,5 Hz und 43,0 Hz Kästen verwischen. Der Prozess „Phasenauspacken“ wird demnach angewendet, um die exakten Frequenzkomponenten zu erreichen, indem es die Signalphasen-Angelpunkte jedes Frequenzkastens mit den Phasen-Angelpunkten der zugehörigen Kästen des vorher verarbeiteten FFT-Fensters abgleicht und dann diese Unterschiede der Phasen-Angelpunkte verwendet, um die frequenzextakten Anpassungen für die festgelegten Frequenzkästen des FFT zu berechnen. Schauen Sie Stufe 3 in Abbildung 2 an. Die Synthese-Stufe ist grundsätzlich die Umkehrung der Analyse-Stufe. Während der Synthese wird ein ähnlicher, aber umgekehrter Vorgang zum Phasenauspacken durchgeführt. Zuerst werden exakte Frequenz/Amplituden-Informationen zu kastenbasiertem Phasen/Amplituden-Format umgewandelt, wird erneut jeden Phasenangelpunkt der Frequenzkästen abgeglichen, so dass sie gut mit den Phasenangelpunkten früher verarbeiteter Soundsätzen justiert sind. Dann ist das Ergebnis umgekehrt-FFT't und wird mit dem früher verarbeiteten Sounddatenstrom kombiniert.

Zeit/Pitch-Skalierung mit Phase Vocoder

Pitch-Verschiebung kann einfach erreicht werden, indem man die exakte Frequenz/Amplituden-Information mit einer passenden Rate zwischen den Analyse- und Synthesestufen dehnt. Erweiterung kann für das Dehnen verwendet werden, aber sogar dehnen, indem man das stärkste von den nächsten Signalwerten als solche nimmt, erzeugt ein ausführbares Ergebnis. Sehen Sie sich Stufe 4 in Abbildung 2 an. Zeitskalierung kann erreicht werden, indem man die Synthesestufe so modifiziert, dass die synthetisierten Verarbeitungssätze zurück auf den Ergebnissound mit verschiedenen, überlappenden Staffelungen während der Analysestufe angewendet werden. Dies ermöglicht längere oder kürzere Tondauer als ursprünglich mit ähnlichen Frequenzcharakteristika aufzubauen.

Praktische Hinweise

Um eine hohe Soundqualität zu erreichen, wird eine große FFT-Umrechnungsgröße wie 4096 oder 8192 benötigt. Damit das Phasenauspacken gut arbeitet, müssen auch die FFT-Fenster miteinander in einer passend hohen Rate überlappen, in der Praxis mindestens vier Mal. Wenn die Zeitskalierung, die die Zeitdauer während der Synthesestufe ausweitet, gewünscht ist, kann sogar eine höhere Überlappungsrate notwendig sein, da die Zeitskalierung während der Synthese die Überlappungsschritte dehnt und somit die Überlappungsrate der Ausgabe effektiv reduziert. Es ist offensichtlich, dass Phase Vocoder eine rechenbetont schwere Aufgabe ist. Neben der großen FFT-Größe und mehrmaligem Überlappen, umfasst das Phasenauspacken während der Analysestufe die Berechnung des Phasenangelpunktes und der Amplitude als Arkustangens und absoluten Wert von komplex-gewerteten Zahlen, ebenso wie die Angleichung der akkumulierten Phasenangelpunkte im Bereich [-π..π]. Die Synthesestufe bezieht gleichermaßen die Berechnung trigonometrischer sin/cos-Funktionen ein, um den Phasenangelpunkt/Amplitude zurück in komplexe Notation zu verwandeln. Diese Vorgänge erzwingen praktisch die Benutzung von Fließkommaarithmetik, machen „nur Integer“-Implementationen undurchführbar. Wenn nur Zeitskalierung ohne simultanes Pitch-Verschieben gewollt ist, dann ist es möglich, die Analyse- und Synthesestufen teilweise zu kombinieren, einige Schritte des Phasenauspakken-Vorgangs wegzulassen und den Analyse-Synthese Phasenangelpunkt-Abgleich direkt durchzuführen, unter Benutzung der komplex-gewerteten Phasen-/Amplituden-/Frequenzwerte, die von FFT erzeugt werden, indem die komplex-gewerteten Phasenangelpunkte mit passenen komplex-gewerteten Multiplikationen rotiert werden. Dies beseitigt den Bedarf nach trigonomischen Funktionen, die anderenfalls für die Analyse- und Synthesestufen benötigt gewesen wären und somit bietet es eine große Verbesserung der Kalkulationseffizienz und ermöglicht ebenso weitere Routinenoptimierungen. Bei der Verarbeitung von Stereo oder anderem Multikanalsound, muss auf die Aufrechterhaltung der Phasensynchronisation zwischen den Kanälen geachtet werden, da der Stereo oder räumliche Effekt anderenfalls ruiniert ist. In der Praxis müssen alle Kanäle zusammen verarbeitet werden, so dass die Phasenkomponenten der selben Frequenzen im gleichen Umfang für alle Kanäle erweitert werden. Zum Beispiel bei den Phasenmo-

SIMD

Single-Instruction-Multiple-Destination oder SIMD bedeutet Instruktionen auf Prozessorebene, die die selbe Funktion für mehrere Datenelemente mit einer einzigen Instruktionen ausführen. Moderne Prozessoren unterstützen SIMD-Instruktionen als Erweiterungen, um die Leistung bei rechenintensiven Routinen zu optimieren. Beispiele für die SIMD-Instruktionssätze sind MMX- und SSE-Erweiterungen der X86-Prozessoren, die in PCs verwendet werden und AltiVec von PowerPPC, die in Mac-Computern verwendet werden. Ein Beispiel für eine SIMD-Instruktion ist der Vorgang der Multiplikation von vier parallelen Zahlen mit vier anderen Zahlen, die vier parallele Werten des Multiplikationsergebnisses erzeugen, demnach bietet das eine nominal vierfach schnellere Leistung als die Verwendung von vier aufeinander folgenden Multiplikationsanweisungen. Doch aufgrund anderes Routine-Overhead ist eine so hohe SIMD-Beschleunigung in der Praxis nicht erreichbar, aber eine Beschleunigung mit einem Faktor von zwei oder drei ist oft sehr gut möglich. SIMD-Optimierungen sind für Routinen passend, die es erlauben, mehrere Berechnungen zur gleichen Zeit durchzuführen, so wie Vektor- und Matrix-Berechnungen. Dennoch, parallele Operationen in Routinen, die in normalen Programmiersprachen geschrieben sind, ist nicht einfach und auch heute noch können Kompiler nicht guten automatischen SIMD-Code erzeugen, der Programmierer muss dem Kompiler helfen, indem er von Hand die Routinen unter Verwendung der SIMD-kompatiblen Syntaxerweiterungen schreibt.

difikationsschritten während der Analyse- und Synthesestufen, wird für jede Frequenzkomponente der Kanal ausgewählt, der die stärkste Amplitude hat und die Phasenänderung dieses einzelnen Kanals wird dann auf alle anderen Soundkanäle gleichzeitig angewendet. Die Verarbeitungslatenz des Phase Vocoder Algorithmus, wieder ein wichtiger Faktor bei Echtzeitverarbeitung und in Applikationen mit strengen Soundsynchronisierungsbedingungen, hängt von der FFT-Größe und dem Überlappungsfaktor ab. Benutzt man eine FFT-Größe von 4096 und ein viermaliges Überlappen bei Sounddaten, die mit 44100 Hz abgetastet werden, ist die Verarbeitungslatenz bei etwa 116 Millisekunden.

Artefakte

Der Vorteil von Zeit/Pitch-Skalierung basierend auf dem Phase Vocoder ist der, dass es keine ähnlich widerhallenden Artefakte wie SOLA erzeugt. Dennoch ist Phase Vocoder keinesfalls frei von Artefakten, sondern die Änderung von Pitch oder Zeit mit Phase Vocoder ergibt größere Phasenverzerrungen. Diese Phasenverzerrungen kommen daher, dass der Phase Vocoder die Phasen jeder Frequenzkomponente abgleicht, so dass sie gut mit den Phasen der gleichen Frequenz des zuvor verarbeiteten Slots zusammenpassen. Da die Signalwellenlänge linear von der Frequenz abhängt, führt die Änderung der Pitch- oder Zeitskalierungen zu verschiedenen Phasenmodifikationsraten bei unterschiedlichen Frequenzen. In der Praxis führt der Betrieb von Pitch- oder Zeitskalierungen mit Phase Vocoder für einen Zeitraum von mehr als ein paar Sekunden zu praktisch zufälligen Signalphasen. Dieser Verlust von Phasenkonsistenz verursacht dumpfen Sound, welcher besonders in scharfen Sounds bemerkbar ist, die von Instrumenten mit einer schnellen Anfallszeit wie Schlaginstrumente, Klavier usw. erzeugt werden. Dies ist ein Hauptproblem im Falle, dass klare, hochwertige Soundqualität für die Musik, die solche Instrumente einbindet, gewollt war. Das Problem der Phasenstimmigkeit ist schwer zu lösen.

Eine Annäherung ist, die Frequenzskala in Gruppen von mehreren benachbarten Frequenzen zu teilen und die Phasen in der selben Gruppe zusammenzuschliessen, so dass die Phasenstimmigkeit zumindest lokal behalten wird. Obwohl es hilft, benötigt sogar diese Annäherung gelegentlich eine Zurücksetzung aller Gruppenphasen, um zu verhindern, dass die Frequenzgruppen zu weit von der ursprünglichen Position abweichen, und solche Phasenzurücksetzungen erzeugen wieder hörbare Artefakte. Eine andere Störung, die von Phase Vocoder verursacht wird, ist der unebene Frequenzbereich, so dass schmale Frequenzbänder über die Spektrumsspanne stark gedämpft werden, so wie das höhere Ende des Frequenzbereichs gedämpft wird. Diese Frequenzbereichsverzerrungen können ein Problem sein oder auch nicht, davon abhängig, welche Art von Sound verarbeitet wird.

Fazit

Der SOLA-Ansatz ermöglicht eine recht einfache und effiziente Zeit/Pitch-Skalierungsimplementation, die gut für Zeit/Pitch-Änderungen in kleinem Umfang arbeitet und kann auf alle Arten von Sound angewendet werden, von menschlicher Sprache zu fertig produzierter Musik. Die Soundqualität wird jedoch für Tempo/Pitch-Modifikationen in größerem Umfang unvorteilhaft, mit Skalierungsraten, die einige zehn Prozent überschreiten und ein widerhallendes Artefakt in den Sound einführen. Die Verarbeitung von Stereosound in CD-Qualität zur Echtzeit mit SOLA erfordert einen 100 Mhz Prozessor mit einer Implementation der Standard-C-Sprache, was bedeutet, dass Echtzeit-SOLA-Verarbeitung mit modernen Pcs und sogar mit portablen Geräten wie PDAs gut durchführbar ist. Der Kreuzverbindungsalgorithmus, der in SOLA verwendet wird, ist sehr passend für CPU-spezifische SIMD-Optimierungen, die die Verminderung der CPU-Verwendung um einen Faktor von zwei bis drei ermöglicht. SOLA kann auch, falls nötig, unter Verwendung von Integer-Arithmetik implementiert werden, was es auch für Prozessoren ohne eingebaute Fließkomma-Unterstützung möglich macht.

Verglichen mit SOLA ist der größte Vorteil der Verwendung von Phase Vocoder zur Zeit/Pitch-Skalierung der, dass es keine ähnlich widerhallenden Artefakte wie SOLA erzeugt, sogar bei Zeit/Pitch-Modifikationen in großem Umfang. Jedoch erzeugt Phase Vocoder andere Artefakte, am meisten bemerkbar ist der dumpfe Sound, der von dem Verlust der Phasensynchronisation und ungleichmäßigen Frequenzbereichen verursacht wird. Phase Vocoder ist gut geeignet, um unabhängige Soundkanalinformationen mit einem einzigen oder wenigen Instrumenten oder menschliche Sprache oder Gesang zu verarbeiten, aber es könnte nicht das Beste sein, um hochwertige Musik zu verarbeiten, die bereits produziert und ins endgültige Aufnahmeformat geschmischt ist. Phase Vocoder ist rechenintensiv, komplexe Berechnungen umfassend und erzwingt daher praktisch die Verwendung von Fließkommaarithmetik. Hochwertige Phase Vocoder-Implementation benötigt etwa eine Größenordnung mehr Rechenleistung als der SOLA Algorithmus. Wenn man nur den Zeitskalierungseffekt anwendet, der folglich erlaubt, einige der zeitraubendsten Stufen auszulassen und manuell abgestimmte Prozessor-Niveau SIMD-Optimierungen verwendet, würde die Verarbeitung von Stereosound in CD-Qualität etwa 500 Mhz auf einem Prozessor vom Level eines Pentium III brauchen.

Weitergehende Informationen

Für eine effiziente open-source Implementierung von SOLA-basierten Zeit-/Pitch-Skalierungsroutinen werfen Sie bitte einen Blick auf die SoundTouch-Bibliothek unter *http://www.surina.net/soundtouch*. Für weitere Information über die Zeit-/Pitch-Skalierungsmodifikationen, die auf Phase Vocoder basieren, zeigt das Dokument *Improved Phase Vocoder Time-Scale modification of Audio* (Laroche, Dolson, IEEE transactions on speech and audio processing, Vol. 7, No. 3, May 1999) (Deutsch: *Verbesserte Phase-Vocoder Zeitskalierungsmodifikation von Audio* (Laroche, Dolson, IEEE Abhandlungen über Sprach- und Audioverarbeitung, Vol. 7, Nr. 3, Mai 1999) – keine Informationen, ob das Dokument in deutsch verfügbar ist) das Konzept von Phase Vocoder und bietet eine Erörterung des Punktes Phasenstimmigkeit. ■