データ制作・本ページ執筆：293 (a.k.a hex125)

はじめに

これはニコニコ動画で公開した「NRTDRV（＋ CSM 音声合成）で「もってけ！セーラーふく」を作ってみた」で行なった音声合成法の解説です。CSM 音声合成用のデータを作成するためのプログラムを公開できれば良いのですが、諸々の事情によりちょっとできそうにありません。そこで、考え方や技術などについて書きたいと思います。

動画を見ていただければ判るとおり、一応、それなりの成果は得られました。しかし、これはいろいろ手を動かしてみて得られた結果をまとめたもので、理論的に正しいのかなんて全く検証していませんし、このテキストも正しい事を書いている保証はありません。プログラムのミスもかなりやらかしましたし、よく分からないまま進めたところもあり、極めて主観的に判断したところもありで、かなりテキトーにやってたことが判るかもしれませんが（笑）このテキストが何かのヒントになれば嬉しく思います。

なお、CSM 音声合成用データの作成には Excel+VBA を使用しました。その中で、露本伊佐男氏作の ExcelVBA 用高速フーリエ変換 (FFT) ルーチン、および新山（へろぱ）氏作の VB6 用ファイルダイアログクラスを使用させて頂きました。この場を借りて御礼申し上げます。

そして、以下のサイトに記述されている内容が大変参考になりました。ありがとうございました。本稿と合わせて御覧になることを強くお勧めします。

- FM 音源 CSM 音声合成（Bookworm's Library）
- FFT とは？～本当は正しくない FFT の周波数特性～（nabe の雑記帳）

CSM 音声合成とは？

CSM 音声合成は音声符号化の一種で、元の音声を複数のサイン波（＝正弦波）の組み合わせで再現する手法です。CSM は Composite Sinusoidal Modeling の略で「複合正弦波モデル化」という意味です。

「どんな連続波形も三角関数の足し算で表せる」という理論があります。この理論を元に、瞬間的な音声波形をサイン波に分解します（周波数解析）。強く現れている音程のサイン波を適切な音量で複数同時に再生すれば、元の波形と似た波形を再現できます。

この音声合成法に必要な要件は「任意の音程・音量のサイン波をできるだけ多く出力できること」だけです。FM 音源はサイン波のカタマリのような音源ですから、この音声合成法と、とても相性が良いと言えるでしょう。事実、YAMAHA の FM 音源チップには CSM 音声合成をサポートするような機能が存在しています。

CSM 音声合成モード

ほとんどの YAMAHA 製の FM 音源には、内蔵のタイマがオーバーフローしたときに全チャンネル

（もしくは一部チャンネル）を即座にキーオン / キーオフする機能があります。YAMAHA はこの機能を「CSM モード」もしくは「CSM 音声合成モード (CSM Speech Synthesis Mode)」と呼んでいます。

開発者は「音の出力を定期的に確実に更新するから、適切に出音の各パラメータを設定してね！じゃ、後はよろしく！」とでも言いたかったのでしょう。どちらかと言えば、これは CSM 音声合成そのものではなく、CSM 音声合成時に必要な待ち時間を管理するための機能と思ったほうが良いでしょう。機能としてはそれだけなので、「CSM 音声合成モードを使えば誰でも音声合成ができる！」というわけではありませんし、「CSM 音声合成モードがないから、CSM 音声合成はできない」ということもありません。

いずれにしても音声合成用のデータは自前で用意する必要があります。また、自前で時間管理を行い適切にサイン波が発音できれば、この方式による音声合成自体は実現可能ですから、CSM 音声合成モードを使わなければならない必然性もありません。事実、本稿の解説も CSM モードは使用していません。NRTDRV が管理している時間単位で処理をしています。

X1 には FM 音源タイマ割り込みがないため、CSM モードを単純利用できません。プログラムを書いて、ムリヤリ使用することは可能だと思われますが、現時点ではそのような仕組みは NRTDRV 上に実装されていません。

> ※ややこしいですが、本稿では「CSM 音声合成モード＝ FM 音源上に搭載されている機能」「CSM 音声合成＝複数のサイン波を同時に出力して元波形を再現する方法」を指しています。明確に使い分けていますが、ココを混同するとワケが分からなくなると思いますので、どうかそのようにお読みください ^^;

■（参考資料）各 FM 音源の CSM モードレジスタ

YM2151(OPM)
　14H bit7
YM2203(OPN), YM2608(OPNA)
　27H bit7/bit6(00= 通常 /01= 効果音モード /10=CSM 音声合成モード)
YM3812(OPL2), Y8950(MSX-AUDIO)
　08H bit7
YM2413B(OPLL)
　確認できず

使用例

最も著名なのは、やはりなんといっても、一連のゲームアーツ制作の PC-8801 用ゲームソフトでしょう。

シルフィード (1986 年) にはじまり、ぎゅわんぶらあ自己中心派 (1987 年)・ゼリアード (1987 年)・ヴェイグス (1988 年) といったゲームで、喋りまくっています。後期は、PC-8801 以外の機種でも FM 音源搭載機種ならば、積極的に CSM 音声合成を使用しています（X1turbo 版のゼリアード (1988 年) や MSX 版のぎゅあんぶらあ自己中心派（1988 年・要 FM-PAC）など）。

もっとも、『喋りまくっている』と言うと語弊があるかもしれませんが・・・「ザカリテ」や「し

めさば」がキーワードでしょうか（笑） もし、ご存じない方は、YouTube 等に動画がありますので、一度ご覧になってみてください。残念ながら、筆者は当時あまり耳にする機会がなかったのですが、それでも店頭デモ等でオープニングに必ず流れる「Presented by GAMEARTS」の音声合成はよく聞きましたし、強いインパクトを受けたものでした。今聞いても、その独特の質感は面白いです。

これらの音声合成を実現していたのは、ゲームアーツ社内で三橋正邦氏が開発した「CSM トーキングシステム」です。ゲームアーツ製のゲーム以外ではほとんどお目にかからなかった手法でしたが、三橋氏の手腕の賜物だったわけですね。

OPM での使用例

実はあまり確認できません。X1turbo 版の「ゼリアード」（ゲームアーツ）くらいでしょうか。

OPM が標準で搭載されていたパソコンは事実上、X1 と X68000 の二機種しかないといって良いでしょう。X1 においては前述の通り、OPM のタイマ割り込みがないため、CSM 音声合成モードは非常に扱いづらいものになっています。X68000 においては ADPCM が搭載されていたため、単にしゃべらせるだけなら ADPCM を利用した方が断然楽でした。

マイナーなところでは MSX(SFG-01) や PC-8801(響) への搭載例もあります。MSX ではシステムに喋らせていたという情報もありますが、残念ながら筆者は未聴です。また、アーケードゲームでの使用実績はかなりの数に登りますが、X68000 と同様の理由で、CSM 音声合成はおそらくほとんど使われてなかったのではないかと考えられます。

利点と欠点

低リソース・低負荷で再生が可能

当時の 8bit パソコンの性能では PCM の処理は容量的にも処理能力的にも厳しいですが、CSM 音声合成は余裕を持って処理できます。ただし補足すると、これは PCM データと比較した場合の話で、単純な演奏データとして見るとそれなりにメモリ喰らいです。「もってけ～」の場合、容量的に TV サイズ版しか作れませんでした。おおよその内訳は、システム及びドライバ部 (拡張用予約を含む) が 16kbyte、バックトラック演奏用データが 8kbyte 弱、CSM 用データが 40kbyte ほどです。

データ作成に時間がかかる

離散フーリエ変換、およびデータの評価はかなり時間のかかる処理です。「シルフィード 100 の秘密」で語られていますが、当時は PC-9801(10MHz) で一秒解析するのに数時間かかったそうです。現在は非力なパソコンでも Excel で数十秒の解析にものの数分～数十分で済みます。良い時代になったものです。

元になる音声波形データが必要

ゼロから作り上げることはできません。特殊な形式ですが「変換」の一種なので、元ネタが無いとなにもできないという、PCM と同じ欠点は引き継いでしまいます。ただし、周波数解析自体が PCM の各種エフェクトに使われる技術でもあります。さまざまな加工ができる余地があり、いろんな可能性を秘めています。

変換元のサンプルによってはまともな音声にならない

人の声なら何でも良いというわけでもなく、大勢の人間がワイワイ騒いでいるようなものや、深いリバーブがかかったもの、ヴォコーダーボイスなどは再現性が非常に悪く、はっきり言っ

てなにがなんだかわからないものになります。複雑な波形の再現には向かない方法だと言えるでしょう。

原著論文

古くからある技術ですが、一体いつごろ確立されたものなのか疑問に思ったので調べてみました。日本国内ではどうやらこの論文が原点のようです。ちなみに筆者は未読です。機会があったら読んでみたいですが、理解できるのかしら。。。

> 嵯峨山茂樹・板倉文忠 (1979 年) 「複合正弦波による音声合成」
> http://hil.t.u-tokyo.ac.jp/~sagayama/

フーリエ変換

素人がフーリエ変換についてあーだこーだ語るのは荷が重いので、具体的な計算方法や処理の仕方については、解説サイトが結構な数ありますので、ググってそちらを参照してください。ここでは概略と要点の解説にとどめておきます。

フーリエ変換を行うと、以下のイメージのように、時間軸での物理量データを周波数軸の強弱データに変換することができます。どの周波数帯の成分が強く現れているのか、一目瞭然になります。

離散フーリエ変換 (DFT) と高速フーリエ変換 (FFT)

本来のフーリエ変換はアナログ波の単一パルスに対して行う処理です。しかし、コンピュータ上で扱えるデータは標本化されたサンプリングデータです。これは連続していない飛び飛びの（＝離散している）データです。コンピュータ上で処理する場合は、離散フーリエ変換 (DFT:Discrete Fourier Transform) を用います。

DFT は計算量が多いため非常に時間がかかります。そこで実際には高速フーリエ変換 (FFT:Fast Fourier Transform) を使用することになると思います。FFT と DFT は基本的に同じ処理なので、計算結果は（誤差を除けば）同じになります。唯一の制限が計算できる波形の長さが 2 の n 乗に限定されてしまいますが、DFT に比べて非常に高速に演算が行えるようになります。

窓関数

フーリエ変換は暗黙のうちに「処理する波形を一周期とした、無限に続く波形」と見なして計算を行っています。この時、始点と終点の値があまりにもかけ離れていたりするなど、音としての連続性に不備があると、本来は存在しない変化が発生してしまい、都合がよろしくありません。

この対策として、自然な形で始点と終点がゼロになるように補正をします。この補正用関数は窓関数 (Window Function) と呼ばれています。（・・・厳密に言うと嘘っパチな解説かもしれません。でも、この認識で問題ないと思います。）目的によって様々な窓関数が使われているそうです。

いろいろテストしてみたのですが、窓関数あり・なしで得られた結果を実際に聞いてみても、筆者には違いがよくわかりませんでした。窓関数を使うと音が丸くなったような感じがする程度

の違いは感じましたが・・・今回は一応クリティカルな場合を想定してハミング窓を採用しました。さらにちなみに窓関数なし（＝なにもしない窓関数）を「矩形窓」と言うそうです。

それなりのキーワードは書きましたので、興味のある方はグッてくださいませ ^-^;

この辺りは「そーゆーもんだ」とあまり深く考えずに居たほうが良いと思います。筆者自身も厳密に理解しているわけではありませんし、ばっきりいってブラックボックスと割りきって使いました。実際の運用も、霧本伊佐男氏の ExcelVBA 用高速 FFT ルーチンの解説を理解するだけで必要十分だと思います。

キッカケは「Speaking Piano（喋るピアノ）」

以上で述べたように、OPM で CSM 音声合成は可能だと想像できるものの、具体的にどうすれば良いのかさっぱり分かりませんでした。

ところが YouTube で公開された「Speaking Piano（喋るピアノ）」を見たことで、状況が一変しました。

有名な動画だと思うのでご存じの方も多いでしょう。英語もドイツ語もわからないのですが、大変驚きました。フーリエ変換による周波数解析を行ったものだというのはわかったのですが・・・

・平均律で出力しても十分声に聞こえる
・ピアノの鍵盤を同時に押している個数は 10 音強程度？で極端に多くはない
・厳密なサイン波ではなくピアノの音色でも構わない

これらの点が驚きでした。特に平均律で再現可能という点は、OPM での再現が可能だと予感させてくれました。というのも、OPN 系 /OPL 系の FM 音源は F-Number(周波数値) で音程を指定しますが、OPM は KeyCode(鍵盤値) 指定なのです。ならば NRTDRV のスペックで十分に再生可能ではないのか？と考え、実行してみることにしました。

OPM で CSM 音声合成・その１（1slot のみを使用した音声合成）

まず、Speaking Piano で行っているであろう手法をそのまま模倣し、1slot のみを使用したサイン波を使って 16 チャンネルの鍵盤を叩き、同等の音声合成を行うことを第一目標に掲げました。

処理の流れの概略は以下のとおりです。

1.（前段階処理）PCM を処理時間単位で分割する
2. 分割した PCM データを離散フーリエ変換 (DFT) にかける
3.HiPass/LoPass フィルタ等をかける
4. 計算結果を評価する
5. 上記の処理を PCM データの終わりまで繰り返す

PCM を処理時間単位で分割する

CSM モードを使用しない場合はドライバの時間管理を利用します。ドライバの最短処理時間をテンポ･タイムベース、そして PCM サンプルのサンプリングレートから算出し、実時間に相当す

る長さで PCM を分割します。計算式は以下のとおりです。

Rate ÷ (BPM ÷ 60) ÷ TimeBase × Step

仮に作る曲のテンポ (BPM) が 120・ドライバの四分音符分解能 (TimeBase) が 48・用意した PCM データのサンプリングレート (Rate) が 44.1kHz・ドライバ分解能の 7Step ごとに処理する場合は 44100 ÷ (120 ÷ 60) ÷ 48 × 7 ＝ 3215.625 となります。この場合、PCM データを 3215 個もしくは 3216 個を読み込んで、その単位ごとにフーリエ変換および評価を行ないます。

ちなみに WAV はヘッダ等がくっついていてちょっと扱いづらいので、筆者は生の PCM に変換してから処理をしました。WAV → PCM のコンバートには run68.exe と、NOZ. 氏作の pcm3pcm.x を使用しました。以前から MDX の作成などでもお世話になっています。ありがとうございます。

CSM モードを使用する場合は、割り込みタイミングの実時間ごとに分割します。

分割した PCM データを離散フーリエ変換 (DFT) にかける ＆ HiPass/LoPass フィルタ等をかける

処理時間短縮のため、高速フーリエ変換 (FFT) を使用します。この変換で周波数値に対する出力量が求められます。

高速フーリエ変換を使った場合、得られるデータは 2 の n 乗個になります。最終データが PCM のサンプリングレートの周波数になるので、x 個目のデータが示すおおよその周波数は「(x ÷ FFT のサイズ) × サンプリングレート」で算出できます。

結果は左右対称になります。意味のあるデータは半分だけです。また、低周波数帯 / 高周波数帯は窓関数等の影響で意味のないデータになることが多いようです。人の声の再生が目的ですので、適当な範囲のデータ以外を削除することはそれなりに意味がある・・・はずです。

今回は HiPass=400Hz・LoPass=7000Hz で処理した・・・と思います（記憶が曖昧・・・）。この数値も特に理由もなく、適当に決めたものです。

計算結果を評価する

ココが肝です。

まず評価する前に、離散フーリエ変換で得られた周波数値を、鍵盤の持つ周波数値と比較し、最も近似の周波数値にあてはめて、出力量を集計し直します。

離散フーリエ変換の結果、厳密に周波数成分が解析できるわけではありません。出力結果の周波数値も離散しているので、「このへんの周波数帯がこのくらい強く現れてます」ということしかわかりません。（詳しくは FFT とは？～本当は正しくない FFT の周波数特性～を参照してください。）そこで、改めて「鍵盤の周波数値」を基準に再集計します。

これで、おおよその周波数値に対する絶対量が、鍵盤に対する絶対量に置き換わります。出力量

の多いデータを上位から必要個数取得して、鍵盤データと音量データを保存します。

ちなみに筆者はこの時の音量値の扱いが未だによくわかっていません。音量は対数変化するので、アレコレいじってなんとなく変換できた気がする・・・という状態にはできたのですが・・・この部分は識者の意見を伺いたいところです ^-^;;;

■結果例

http://nrtdrv.sakura.ne.jp/mp3/csmtest_20100522_16.mp3

これは 1slot × 16ch で処理したものです（窓関数なし･ほぼ LoPass/HiPass を適用せず）。なお、申し訳ありませんが、音量に注意してください（汗）。かなりデカイです（汗）

OPM で CSM 音声合成・その２（4slot を使用した音声合成）

その１で取った方法だと、発声だけで全チャンネルを消費してしまいます。RPG 等のゲームの演出で、音声のみ出力したい場合には使える方法ですが、ヴォーカル表現への転用を考えると音楽との同期再生ができなければ実用とは言えません。また、使用スロットが 4 個中 1 個と、非常に勿体無い使い方をしています。そこで 4 つのスロット全てを有効的に使う方法はないか考えてみました。

倍音構成に再集計する

その１とほとんど同じ処理を行ないますが、決定的に違うのは「計算結果を評価する」部分です。その１では鍵盤の周波数に対して再集計しましたが、これを OPM で表現可能な倍音設定の周波数に変更します。

OPM は MUL と DT2 の組み合わせで 64 段階の倍音を表現できます。MUL は倍音、DT2 は倍音の倍率自体の設定項ですから、MUL(16 段階) × DT2(4 段階)=64 段階となります。

上図の音色を使用した場合、o4a(440Hz) の音を出す指示を行っても、実際に出力されるのは o3a(220Hz) になります。アルゴリズム 7 で OP1 のみ有効・MUL=0/DT2=0 なことに注目してください。

オペレータは 4 個ありますから、鍵盤の音程を固定して MUL と DT2 を操作するだけで、64 個中 4 個の別の周波数の音を 1ch で出力できます。この 64 個の周波数は不定ですが、叩く鍵盤の音程を決めた時点で、出力可能な 64 個の周波数が決まります。離散フーリエ変換の結果を、この 64 個の周波数を基準に再集計します。

■結果例

その１では、鍵盤の音程＝平均律に従うので、なんとなく多少音痴でも補正されるんじゃないか？という気がしますが、この方法は平均律を完全に無視しています。ですから、とても音痴になる可能性があると予測できます。実際その通りで、試してみると以下のサンプルのようにかなり怪しげな音声になりました。

http://nrtdrv.sakura.ne.jp/mp3/csmtest_20100523.mp3

これは 4slot × 4ch で全チャンネルで o4c を叩いたものです（窓関数なし・ほぼ LoPass/HiPass を適用せず)。申し訳ありませんが、こちらも音量注意です（汗）

明らかに元にはない成分が目立った音声になっています。

複数チャンネルを使用して発音可能な倍音を増やす

周波数帯をイビツな形で 64 段階に分けただけではクオリティ的にはまだまだ問題があるようです。そこで、複数チャンネルを利用し、叩く鍵盤をズラして、発音可能な倍音成分を増やしてみます。

上図は､平均律と OPM で出力可能な倍音の周波数値の対比図です。平均律は対数変化しているのに対し、倍音はわりと直線的に変化しています。このことからも、倍音利用ではその１と同等の音声合成はできないことがわかります。

倍音は 1ch のみ使用した場合も 2ch 使用した場合もあまり変化自体は変わりません。しかし 1ch のみでは 64 段階ですが､2ch 使用すると 128 段階になるので､2ch を使用した場合は､よりきめ細かくなっています。

そして､NRTDRV で指示できる鍵盤情報は o0d+ 〜 o8d+ の 97 個です (OPM の場合)。複数チャンネルを使用すると、かなりイビツな形ではあるものの、きめ細かさでは、その１を上回ることがわかります。

ただし､1 つのチャンネルで発音できる周波数は 64 個のままですから､仮に Ach と Bch の 2 つのチャンネルで演奏するとしたらば、Ach でしか出力できない周波数と Bch でしか出力できない周波数が発生します。そこで多少変則的になりますが、基準とするテーブルに「どのチャンネルで出力可能か？」というフラグを持たせ、評価の際にはフラグを参照して、そのチャンネルで発音可能な上位 4 個のデータをそれぞれに取得します。

基本となる鍵盤をどれにするか､また､どれだけ基本となる鍵盤をズラすかは重要な問題ですが､実はあまりテストをしていません。この辺りは最も今後の検証が必要な箇所だと思います。

「もってけ〜」は F major の曲なので､o4f を基本にすると良いだろうと考えていたのですが､実際に試してみると妙な音声になったので､今回は o4c を基準にしました。また､ズラす幅も半音･全音・4 度・5 度を試してみたところ、半音が一番明瞭に思えたので、今回は半音を採用しました。かなりテキトーです（汗）

■結果例

http://nrtdrv.sakura.ne.jp/mp3/csmtest_20100524.mp3

同じく､4slot × 4ch で o4c/o4c+/o4d/o4d+ を叩いたものです(ハミング窓使用･過度に LoPass/HiPass

を適用）

若干怪しさを感じるものの、64 段階で処理したものと比べると差は歴然です。

まとめ

成果物

ニコニコ動画で公開したバージョンの MML です。

- http://nrtdrv.sakura.ne.jp/mml/motteke20100608mml.zip

ところがこの MML には問題がありました。あまりにデータを詰め込みすぎたため、本来使用してはいけない部分も使用していました。具体的には X1 ではメインメモリの &HFF00 ～ &HFFFF までは、IPL 起動時に BIOS が使用するワークエリアとなっているのですが、この部分にまで音楽データが侵食していたため、ディスクから起動できなくなってしまい、まともに実機で演奏できないことになってしまいました。

すーぱーたーぼ氏作の全二重高速シリアル通信モジュール＋リモートブート (hssio) を使用すれば NRTDRV の IPL 起動が不要になるので実行できるのですが(大変お世話になっています。大感謝です！)、やはりディスクから起動できないのは問題なので、2010/06/13 版からは音楽データの最大サイズがワークエリアにかからないよう修正されました。そのためこのままではコンパイルできません。冒頭セリフ部と、音楽部をわけた MML を改めて用意しましたので、コンパイルしてみたい方はこちらを使ってください。

- MML
  - http://nrtdrv.sakura.ne.jp/mml/motteke_opvoice.mml (冒頭セリフ部)
  - http://nrtdrv.sakura.ne.jp/mml/motteke_tvsize.mml (本編)
- MP3
  - http://nrtdrv.sakura.ne.jp/mp3/motteke_opvoice.mp3 (冒頭セリフ部 /hoot 出力)
  - http://nrtdrv.sakura.ne.jp/mp3/motteke_csmonly.mp3 (本編・CSM 音声合成部のみ /hoot 出力)
  - http://nrtdrv.sakura.ne.jp/mp3/motteke_tvsize.mp3 (本編 /hoot 出力)

追記:現在のコンパイラは曲データの最適化が実装されていますので、MML を分割しなくてもコンパイルでき、ディスクからも起動できるようになっています。

考察及び今後の課題

結果例で示したものと比べると、完成版は音声が劣化している・・・と感じると思います。これには以下のような要因があげられると思います。

- 使用できるメモリ容量の都合で、割り込み間隔を長めにとったため
- 同じく、使用できるメモリ容量の都合で、4slot × 2ch の 8 個のサイン波出力に変更したため
- ハニング窓を使用した／ hipass・lopass を過度にかけたため

割り込み間隔は短すぎるとブザーのようになってしまい、長すぎるとわけのわからない音になってしまいます。ある程度の幅は許容してくれるようですが、適切な割り込みタイミングはトライ＆エラーを繰り返して確認するのが良いと思います。ちなみにテストでは 5step（44.1kHz から変換）、完成版では 7step（32kHz から変換）で処理しました。割り込み間隔が長くなると、データ量は少なくて済みますが、変化に乏しくなるので声としての精度は落ちます。この辺のさじ加減を変えられるのはある意味面白いですね。

何個のサイン波を使うべきか・・・正直、OPN などで一般的に語られている 4 個では少なすぎると感じました。8 個あればそれなりに喋ってくれてる感が出てくれるなと感じました。16 個だと確かに明瞭になってくれるのですが、8 個の場合と比べて二倍の精度があるとはちょっと思えませんでした。というわけで、僕的には 8 音を基準に考えるのが良いかなと（極めて主観的に）思います。

CSM 音声合成のデータ量は、PCM と比較すると圧倒的に小さいですが、シーケンスデータと比較するとかなり大きいです。

方法その２は y コマンドや z コマンドを使えば処理そのものは全く問題なく行えます。（y コマンドの場合は）コマンドバイトと OPM のアドレス値と書き換えるデータ値の 3byte、さらにオペレータ数 (4) と TL/MUL/DT の 3 箇所を書き換える必要がありますから、1 回分の書き換えデータに 3 × 4 × 3 ＝ 36byte 必要になります。

このデータ量を減らすために非公開コマンド $04 を作りました。こちらではデータ量が 9byte で済みます（詳しくは解説ページを参照してください）。ただし、このコマンドを使って精度を落としたデータを使っても 1 分強しか喋らせられませんでした。

今回解説した方法その１は、Speaking Piano の手法の模倣で、その２はその変形です。共通している考え方として、フーリエ変換の結果を再集計し、容易に発音できる周波数に集約して出力することが挙げられます。声のようなものを再現したいだけなら、この考え方に問題はなさそうです。

同一の波形であれば、基本的にはフーリエ変換の計算結果は変わりません（窓関数等の影響で若干変化はしますが、それを考えなければ変わらないと思って良いと思います）。ですから、一番の問題は計算結果をどのように評価し、どのデータを取得するかということに尽きると思います。

その１では、音色パラメータの倍音を固定し、叩く鍵盤を変えることで、出力する周波数帯を区別しています。この方法は NRTDRV 以外の音楽制作環境でも、容易に実現できます。特に平均律に従う点は興味深く、多少音痴でも修正してくれそうな気がしますし、安定性もかなり良いように思います。しかし反面、非常に多くチャンネルを消費します。音楽との同期演奏を行う場合は、バックトラック演奏用のデータに割けるチャンネル数が激減してしまいます。

対して、その２は叩く鍵盤を固定し、倍音操作で出力する周波数帯を区別しています。これはOPM の特性を利用したものですから、NRTDRV や MXDRV 等の YM2151 を操作する環境でなければ実現できません。また、平均律を完全に無視しているためか、ピッチが安定しません（この点も、今後模索の必要があるでしょう）。しかし 1 チャンネルで 4 個のサイン波が出せるため、消費するチャンネル数は少なくて済みます。

では、鍵盤と倍音の両方を同時に変更できたら･･･両者の良いトコ取りをして、よりクオリティの良い音声表現ができるかもしれませんが・・・どうやったらそんな評価ができるのでしょうか。。。 どなたか良いアイデアがあれば是非教えていただきたいと思います ^-^;;;

(293)