フーリエ変換を用いたシンセサイザーの試作

箕　原　辰　夫

1．背景

フーリエ（Jean Baptiste Joseph Fourier)[1] は，1796年代数方程式の実数解に関する「フーリエの定理」を発表した。これは，周期関数を正弦（sine 関数）や余弦（cosine 関数）の和を用いて現れるもので，周期関数を解析して，時間関数を周波数の軸に投影できる，所謂「周波数スペクトル」による様々な物理化学的な解析を可能とするものであった。これを「フーリエ級数」と呼ぶ。ただし，実関数だけに留まらず，オイラー（Leonhard Euler)[2] の以下の複素数に関する有名な公式，理工系の大学生であれば誰でも必ず知っている公式，を用いて，複素数も含めて展開する方が簡単に記述できるため，複素数を含めた形で記述されたものを「フーリエ変換」と呼ぶ。

$$e^{i\theta} = \cos(\theta) + i\sin(\theta) \quad \text{あるいは} \quad e^{i2\pi\theta} = \cos(2\pi\theta) + i\sin(2\pi\theta) \qquad (1)$$

フーリエ変換は，周期関数を周波数領域に投影できるため，周期関数，これは複雑な波動関数のことが多いが，その解析のために工学的には幅広く用いられている。また，フーリエ変換の逆を行なう「逆フーリエ変換」は，周波数領域から時間関数を生成できるために，周期的な波動による関数を合成するために，同じように工学的に様々な用途に用いられている。コンピュータ上で扱うためには，波動関数がデジタル化された状態に対して行なわなければならない。なぜならば，様々な録音・録画機器で得られた映像・画像・音声（録音）については，アナログの波動を標本化（sampling）し，量子化（quantization）して，離散化情報，即ちデジタル情報として表現されているからである。そのため，コンピュータ上では，離散化された周期波動関数に対する「離散フーリエ変換・逆変換」[3] が用いられている。これは，フーリエ級数の応用版になっている。

離散フーリエ変換・逆変換によって，多くの信号解析・信号合成が行なわれている。その最たるものは，メディアの劣化圧縮（あるいは不可逆圧縮）である。JPEG や MPEG 等の画像・映像・音声の規格[4]において，離散フーリエ変換・逆変換による圧縮と復号は，データサイズを小さくするために必須であり，その規格名よりも，実際の圧縮方法を示したアルゴリズムの名前，専門用語としては「コーデック」（codec）と呼ばれるが，その

(1)　Wikipedia「フーリエ」の項目を参照（閲覧2009）
(2)　William Dunham,「オイラー入門」, Springer-Verlag 東京，2004.
(3)　高倉葉子,「数値計算の基礎　―解放と誤差―」，コロナ社，2007.
(4)　藤原洋監修，Tech I vol. 4「画像 & 音声圧縮技術のすべて」第 9 版，CQ 出版社，2005.

名前の方が広く知れ渡っている。たとえば，広く音楽配信で用いられている MP3 は，MPEG-1 Audio Layer-3 の省略形である。この頃は，MP3 のメディア品質では満足できないため，AAC（Advance Audio Coding）が普及し始めている。どちらも，離散フーリエ変換の実数形である離散コサイン変換がデータの圧縮に用いられている。

さらに，離散フーリエ変換・逆変換においては，いくつかの高速化のアルゴリズムが考えられており，それらは，総称して「高速フーリエ変換・逆変換」（略称 FFT・IFFT）[5] と呼ばれている。これは，高速フーリエ変換の計算量 $O(N^2)$ を，$O(N\ \log\ N)$ に落とせるために，多くの場合は，離散フーリエ変換は FFT で実装されていることが多い。以上の流れをまとめると，次のようになる。

実数のフーリエ級数　→複素数のフーリエ変換
　　→複素数の離散フーリエ変換→高速フーリエ変換

また，複素数に関わる計算量を減らすために，実数領域において余弦関数を用いた変換が用いられることが多い。これらは，「離散コサイン変換・逆変換」（略称 DCT・IDCT）[6] と呼ばれる。離散コサイン変換にも，高速フーリエ変換のアルゴリズムが適用されることが多い。なお，音声信号の圧縮には，この修正版である「修正離散コサイン変換・逆変換」（略称 MDCT・IMDCT）が用いられることが多く，この変換にも高速フーリエ変換のアルゴリズムの一部が適用可能となっている。

1．1．フーリエ級数[7]

フーリエ級数（Fourier-series transform）は，連続な関数 $x(t)$ が周期 T を有するとき，三角関数を用いて展開したものである。そのため，「フーリエ級数展開」とも呼ばれる。

$$x(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty}\left(a_n \cos\frac{2\pi nt}{T} + b_n \sin\frac{2\pi nt}{T}\right) \qquad (2)$$

係数 a_n および b_n は，次のように求められ，「フーリエ係数」と呼ばれる。

$$\begin{aligned} a_n &= \frac{2}{T}\int_0^{\infty} x(t)\cos\frac{2\pi nt}{T}dt \\ b_n &= \frac{2}{T}\int_0^{\infty} x(t)\sin\frac{2\pi nt}{T}dt \end{aligned} \qquad (3)$$

ただし，これは複素数表示をした方が簡単に表すことができるので，（1）のオイラーの公式を用いて，通常は，次のように複素数の指数関数として展開される。

---

(5)　森正武，名取亮，鳥居達生，岩波講座情報科学-18「数値計算」，岩波書店，1982.

(6)　Wikipedia「離散コサイン変換」の項目を参照（閲覧2009）

(7)　1.1.～1.3. および 1.5. の数式は，高倉葉子「数値解析の基礎」の記述に基づいている。pp.209-223.

$$x(T) = \sum_{n=-\infty}^{\infty} c_n e^{i2\pi nt/T} \qquad (4)$$

このときの係数c_nは，「複素フーリエ係数」と呼ばれる。

$$c_n = \frac{1}{T}\int_0^T x(t)e^{-i2\pi nt/T}dt \qquad (5)$$

（4）および（5）から導かれることは，フーリエ級数では基本周波数$f_1 = 1/T$の整数倍の周波数の周期波動関数が現れる。このため，c_nを順に並べたものを線スペクトルという。

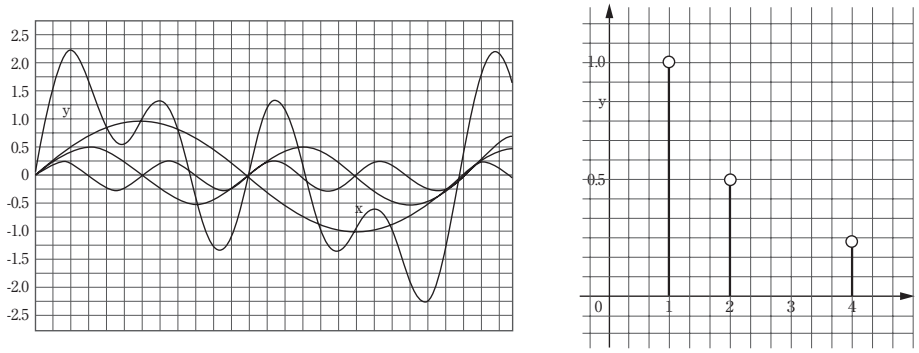

図１　三角関数による合成波形と周波数領域での線スペクトル

１．２．フーリエ変換・逆変換

ここでは，本研究と直接の関係はないものの，フーリエ変換全般に亙る目的は，「周波数と時間軸（あるいは画像では座標軸）との変換」であるということを示すために，フーリエ変換・逆変換の公式だけ紹介する。フーリエ級数において，周期波動関数のみならず，非周期の波動も表すために，周期Ｔを無限大，すなわち$T \to \infty$としたのが，フーリエ変換である。そのため，基本周波数$f_1 = 1/T$は無限小となるので，基本周波数の整数倍であった周波数は，連続量として扱うことが可能になる。$f_1 = \Delta f$とおいて連続量として（4）と（5）を書き換えると，

$$\begin{aligned} x(t) &= \lim_{T\to\infty} \sum_{n=-\infty}^{\infty} \left(\frac{1}{T}\int_0^T x(t')e^{-i2\pi nt'/T}dt'\right)e^{i2\pi nt/T} \\ &= \lim_{T\to\infty} \sum_{f=-\infty}^{\infty} \Delta f\left(\int_0^T x(t')e^{-i2\pi ft'}dt'\right)e^{i2\pi ft} \qquad (7) \\ &= \int_{-\infty}^{\infty}\left(\int_0^{\infty} x(t')e^{-i2\pi ft'}dt'\right)e^{i2\pi ft}df \end{aligned}$$

となり，この式の括弧内を$X(f)$と置くと

$$X(f) = \int_0^{\infty} x(t) e^{-i2\pi ft} dt \qquad (8)$$

$$x(t) = \int_{-\infty}^{\infty} X(f) e^{i2\pi ft} df \qquad (9)$$

となる。（8）はフーリエ変換と呼ばれ，時間軸から周波数領域への変換を行なうものである。（9）は逆フーリエ変換と呼び，周波数領域から，時間軸に変換され，実際の波動を再現するが可能となる。

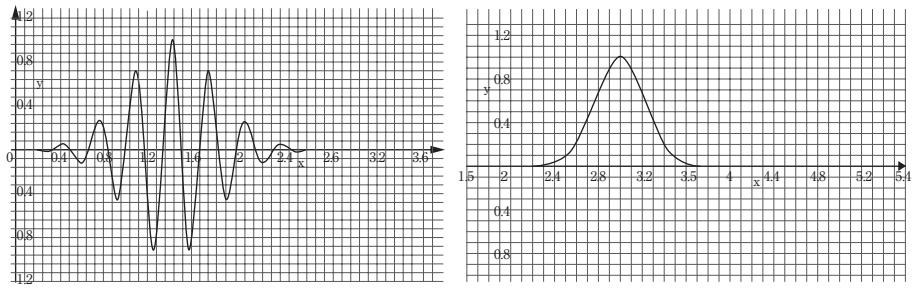

図 2　積分可能な波動関数と，その周波数領域での連続スペクトル

フーリエ変換は，ラプラス変換と共に，理工系の学部では 3 年次ぐらいまでに，その原理や計算方法を履修する基本的な変換方法である。電気電子情報工学系では，（8）および（9）の式は，複素角周波数 $\omega = 2\pi f$ および虚数単位に j を用いて，以下のように表されるのが普通となっている。

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t} dt \qquad (10)$$

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{j\omega t} d\omega \qquad (11)$$

しかしながら，このフーリエ変換では，無限大まで広げて波動の測定値を求めなければならないので，現実的ではない。またフーリエ変換ができる波動関数は，積分可能な関数に限られている。そのため，工学で用いられているのは，決められた有限時間まで行なうフーリエ変換が主体になり，主に時間的に限られた範囲の波動関数の解析に用いられる。アナログのフーリエ変換の応用としては，音声については，オーディオシステムにおいて，スペクトラム・アナライザーのように周波数領域にわけてボリュームを表示する機器や，イコライザーのように，周波数領域によって増幅の比率を変えるようにできるフィルタ(8)として機能するアンプリファイア（増幅器）として，音声信号に使われている。また，高周波である光を含む電磁波など，さまざまな波動においてスペクトル解析に用いられてい

(8)　三上直樹，「はじめて学ぶディジタル・フィルタと高速フーリエ変換」第 2 版，CQ 出版社，2006.

る。連続値とはいえ，サンプリングされる時間範囲が限られていることから，その逆フーリエ変換によって，復号された波動関数は，元の波動関数に比べて劣化している。これは，（8）あるいは（10）の積分の上限と下限が ∞ と $-\infty$ ではなく，0 から T^{ω} までの有限時間になっており，その範囲内の波動関数から周波数解析を行なう。また，それ周波数解析に基づいて，更に，（9）あるいは（11）の逆変換を用いて有限の周波数の範囲内で波動関数を合成することもある。この場合，周波数領域において折り返しをするときに歪みが発生するため，波動関数を合成するときは，その補正をする必要がある。

逆フーリエ変換は，フーリエ合成とも言われているが，いくつかの余弦波（あるいは位相を変えれば正弦波）から，目的の音色に近い音を作り出すために用いられることや，データの転送など用いられることが多い。転送では FM 放送[9]，また，音声を合成するシンセサイザーは，FM 音源として知られており，古くは KORG 社などのアナログシンセサイザーや，ハモンドオルガンあるいはポリフォニック・オルガン（これは非常に限定的な離散コサイン逆変換の応用であるが）などの機器がこの原理で音を合成している。これも複素数領域を使うものと，後で述べる実数領域だけを使うものとがある。

1．3．離散フーリエ変換・逆変換

離散フーリエ変換（Discrete Fourier Transform: DFT）は，マルチメディアにおいて，圧縮技術（codec）の基本的な技法として用いられている。上記のアナログ値における有限区間でのフーリエ変換と同様に，$x(t)$ が区間 0 ～ T^{ω} 以外で 0 であると仮定し，均等な N 回の標本化（サンプリング）を行なう。サンプリング時間の列 $t_j = jT^{\omega}/N$ を仮定すると，

$$X(f_k) = \frac{T^{\omega}}{N} \sum_{j=0}^{N-1} x(t_j) e^{-i2\pi kj/N} \tag{12}$$

となり，(12) から係数 T^{ω}/N を取り除いた $y(f_k) = X(f_k)N/T^{\omega}$ を「離散フーリエ変換」と呼ぶ。T^{ω} は「サンプリング区間」，N/T^{ω} は「サンプリング周波数」と呼ばれる。また更に，$y(f_k) = y_k$，および $x(t_j) = x_j$ と置くと，離散フーリエ変換は，以下のように書き表すことができる。

$$y_k = \sum_{j=0}^{N-1} x_j \omega_N^{kj} \tag{13}$$

(13) の式において，ω_N は「回転因子」(twiddle factor) と呼ばれ，次の (14) で定義される。回転因子は，$\boldsymbol{y}$ と $\boldsymbol{x}$ をベクトル表記した場合，行列としても書き表すことが可能である。

$$\omega_N = e^{-i2\pi/N} \tag{14}$$

$y_k = y(f_k)$，$x_j = x(t_j)$，および回転因子 ω_N を利用して，離散フーリエ逆変換は，以下のように記述することができる。

---

(9) Paul J. Nahin,「オイラー博士の素敵な数式」，邦訳日本評論社，2008.

$$x_j = \frac{1}{N}\sum_{k=0}^{N-1} y_k\,\omega_N^{-jk}\quad (j = 0,1,\cdots,\ N-1) \tag{15}$$

標本化定理から，$f_s = N/T^{\omega}$ と置くと，$f_n = f_s/2$ までの周波数成分を持つ波動が離散フーリエ逆変換によって再現可能となる。f_n は「ナイキスト周波数」と呼ばれる。また，$y_k = y(f_k)$ が複素数になることから，各周波数に関して，振幅スペクトラム，パワー・スペクトラム，および位相角スペクトラムが，それぞれ以下の式によって求めることができる。

$$|y(f_k)| = \sqrt{\mathrm{Re}(y_k)^2+\mathrm{Im}(y_k)^2}\quad |y(f_k)|^2 = \mathrm{Re}(y_k)^2+\mathrm{Im}(y_k)^2\quad \arg(y(f_k)) = \frac{\mathrm{Im}(y_k)}{\mathrm{Re}(y_k)} \tag{16}$$

1．4．離散コサイン変換・逆変換

複素数領域において，虚数軸を含めた計算になると，複素数を計算しなければならず，リアルタイム（実時間）でメディアを合成・解析するためには，かなりの負荷となる。また，実関数を元にした変換においては，エルミート対称[10]になることがわかっている[11]。そのため，離散フーリエ変換を実数の領域だけで行なうのが，離散コサイン変換・逆変換である。この変換は以下のような公式で変換が可能になる。これらメディアの大量のデータサイズの圧縮に用いられることが多く，JPEG，MPEGといった，画像，映像，音声のデジタル表現形式の圧縮方法の，符号化および復号に用いられている。ここでは，DCT-ⅡとDCT-Ⅲ[12]と分類された離散コサイン変換を掲載する。これらは，離散コサイン変換と逆変換を表している。

$$X_k = \sum_{j=0}^{N-1} x_j \cos\left(\frac{\pi}{N}\left(j+\frac{1}{2}\right)k\right) \tag{17}$$

$$x_j = \frac{1}{2}X_0 + \sum_{k=1}^{N-1} X_k \cos\left(\frac{\pi}{N}\left(j+\frac{1}{2}\right)k\right) \tag{18}$$

1．5．高速フーリエ変換

離散フーリエ変換を，回転因子を用いて，ベクトル表記で表した場合，次のような式になる。

$$\boldsymbol{y} = F_N\boldsymbol{x}\ \text{ただし，}\ \boldsymbol{y} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{bmatrix},\ \boldsymbol{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix},\ F_N = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_N^1 & \omega_N^2 & \cdots & \omega_N^{N-1} \\ 1 & \omega_N^2 & \omega_N^4 & \cdots & \omega_N^{2(N-1)} \\ 1 & \vdots & \vdots & & \vdots \\ 1 & \omega_N^{N-1} & \omega_N^{2(N-1)} & \cdots & \omega_N^{(N-1)(N-1)} \end{bmatrix} \tag{19}$$

⑽ 虚数部の大きさが，符号を反対にした領域（共役）において同じになること

⑾ 森正武，名取亮，鳥居達生，岩波講座情報科学-18「数値計算」，岩波書店，1982.

⑿ Wikipedia「離散コサイン変換」の項目を参照（閲覧2009）

この式において，変換行列を計算するのには計算量として，$O(N^2)$ 掛かる。そのため，この計算量を少なくする方式として，高速フーリエ変換（Fast Fourier Transform）のアルゴリズムがいくつか提案されている。主に，時間間引き方式と周波数間引き方式に分かれるが，ここでは時間間引き方式の Cooley-Turkey 型 FFT アルゴリズム[13][14][15]を用いる。このアルゴリズムでは，サンプリングされた値が $N=2^m$ 個あるときに，段数 m で係数を計算していくため，$O(N\ \log_2\ N)$ で計算量が収まるものである。(13) において，x_k を k の偶数列と奇数列で分けると以下のようになる。

$$\begin{aligned} y_k &= \sum_{r=0}^{N/2-1} x_{2r}\omega_N^{(2r)k} + \sum_{r=0}^{N/2-1} x_{2r+1}\omega_N^{(2r+1)k} \\ &= \sum_{r=0}^{N/2-1} x_{2r}\omega_N^{2rk} + \omega_N^k \sum_{r=0}^{N/2-1} x_{2r+1}\omega_N^{2rk} \end{aligned} \tag{20}$$

ここで，以下の等式が成り立つ。

$$\omega_N^2 = e^{-i2\cdot 2\pi/N} = e^{-i2\pi/(N/2)} = \omega_{N/2} \tag{21}$$

この（21）によって，(20) は次のように書き直すことができる。

$$y_k = \sum_{j=0}^{N-1} x_j\omega_N^{kj} = X_k^{(0)} + \omega_N^k X_k^{(1)} \quad (k=0,1,\cdots,\ N-1) \tag{22}$$

$$\left.\begin{aligned} X_{k'}^{(0)} &= \sum_{r=0}^{N/2-1} x_{2r}\omega_{N/2}^{rk'} \quad X_{k'+N/2}^{(0)} = X_{k'}^{(0)} \\ X_{k'}^{(1)} &= \sum_{r=0}^{N/2-1} x_{2r+1}\omega_{N/2}^{rk'} \quad X_{k'+N/2}^{(1)} = X_{k'}^{(1)} \end{aligned}\right\} \ (k'=0,1,\cdots,\ \frac{N}{2}-1) \tag{23}$$

(23) において，$X_{k'}^{(0)}$ および $X_{k'}^{(1)}$ は，データ点数が $N/2$ の離散フーリエ変換となっている。これが第 m 段目の計算になる。これを再帰的に，第 1 段目まで適用すると，すべての値が求められることができる。データ総数が $N=2^m$ であるときに，入力データ x_k の順序を，その添え字を 2 進数表記で表したときに，最上位桁から最下位桁を反対にして並べ替えた結果を，$X_0\,X_1,\ldots,\ X_{N-1}$ と表記すれば，p 段目ではデータ点数が $N_p=2^p$ のグループが $N/N^p=2^{m-p}$ 個あり，各グループの要素は $\omega_{N_p}^{k+N_p/2} = -\omega_{N_p}^k$ を用いて，以下のように偶数列と奇数列に分解される。

$$\left.\begin{aligned} X_k &= X_k^{even} + \omega_{N_p}^k X_k^{odd} \\ X_{k+N_p/2} &= X_k^{even} - \omega_{N_p}^k X_k^{odd} \end{aligned}\right\} \ (k=0,1,\cdots,\ N_p/2-1) \tag{25}$$

これを $p=1$ から始め，$p=m$ になるまで行なえば良い。アルゴリズム的に，高速フー

---

⑬ Wikipedia「高速フーリエ変換」の項目を参照（閲覧2009）

⑭ 表記は，高倉葉子「数値解析の基礎」の記述に基づく。pp.220-223.

⑮ 森正武，名取亮，鳥居達生，岩波講座情報科学-18「数値計算」，岩波書店，1982.

リエ変換の概略を，ビット反転の関数および変換関数を Python のスクリプトを使って表すと，次のようになる。実際のライブラリでも，繰返しを使って計算される。その方が，ベクトル演算ユニットを持つプロセッサにおいては高速に計算できるからである。

```
# 添え字のビットを反転する関数（引数は添え字の数とビット長）
def bitInverse(number, n):
    p = int(math.log(n, 2))
    bi = 0
    for i in range(p):
        base = 2**(p-1-i)          # **演算子はべき乗を示す
        orgbit = number & 1        # & 演算子はビットの AND を示す
        reversebit = base* orgbit
        bi = bi + reversebit
        number >>= 1               # >> 演算子は右シフトを示す
    return bi

# FFT を計算する関数（引数は段数）
def FFT(m):
    global X, N                    # X, N は大域変数とする
    for p in range(1, m+1):
        Np = 2**p
        for i in range(N / Np):
            for j in range(Np / 2):
                theta = j*2*math.pi / Np
                w = complex(math.cos(theta), -math.sin(theta))
                k = Np*i + j
                Xeven = X[k]
                Xodd = X[k+Np/2]
                X[k] = Xeven+w* Xodd
                X[k+Np/2] = Xeven-w*Xodd
```

2．実装

2.1．実装環境

実装は，Macintosh 上の CoreAudio[16][17]の開発環境で行なった。この環境を選んだのは，同様の環境が他のプラットフォームになく，また，開発環境として優れていたからである。特に，CoreAudio では，ここのモジュールをコンポーネントとして開発すればよく，今

---

⑯ CoreAudio Overview, Apple Computer, Inc., Version released in 2009.

⑰ Audio Unit Programming Guide, Apple Computer, Inc., Version released in 2009.

回のモジュールは，CoreAudio の Oscillator として開発した。このモジュールを MIDI のキーボードからピッチを入力したりやスピーカー出力する機能については，他のモジュールあるいはアプリケーションで揃えられるので，シンセサイザーに要求されるすべての機能を単体のアプリケーションとして実装しなくても済むことができた。これらのモジュールやアプリケーションを統合する機能が CoreAudio というフレームワークで Mac OS X 上に用意されており，開発ライセンスさえ取得していれば，SDK（Software Development Kit）を自由にダウンロードできる。また，このフレームワークを用いて，Logic や Cubase といった，Mac OS X 上の統合シーケンサー（Sequencer）環境のアプリケーションが稼働している。そのため，今回作ったモジュールは，それらの環境に組み込むことも可能となっている。

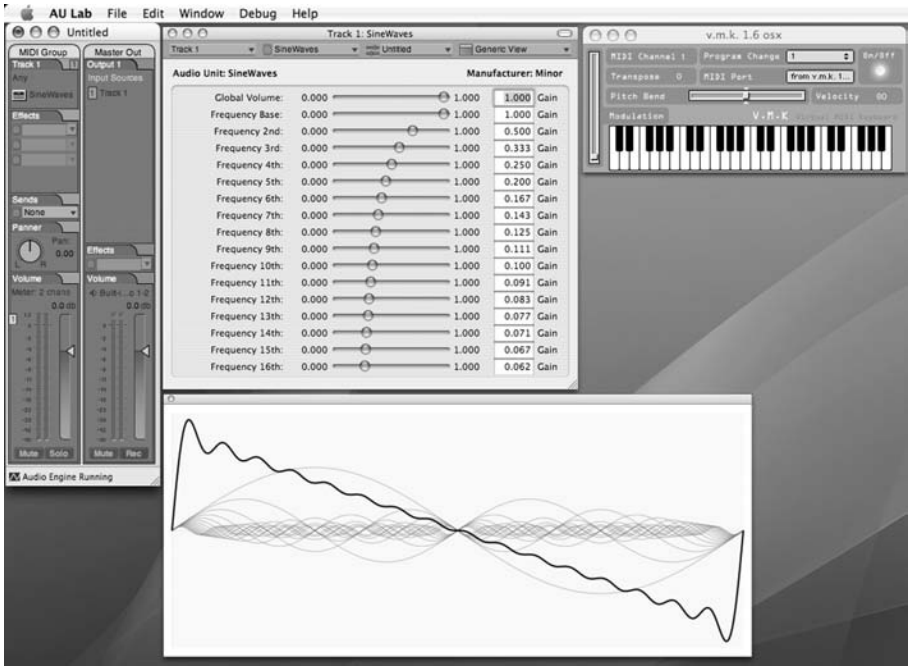

図 3　CoreAudio 上で実装されたモジュール

このスナップショットでは，MIDI の仮想的なキーボードとして，v. m. k. 1.6というフリーソフトウェアを用いている。また，CoreAudio SDK[18]についている AU Lab というソフトウェアを母体として，コンポーネントのテストを行なっている。

2．2．高速フーリエ変換による実装

高速フーリエ変換については，vDSP[19]というライブラリを用いた。このライブラリは，PowerPC G4/G5 のベクトル演算ユニットに調整されて最適化されたものなので，ベクト

---

[18] CoreAudio SDK, version 1.5, Apple Computer, Inc., First released in 2007.

[19] vDSP Library, Apple Computer, Inc., Version released in 2009.

ル演算ユニットの大きさがそれに比べて小さい，Intel Core 2 Duo プロセッサで利用した場合，どの程度遅延がでるのか，評価においてアーキテクチャ的な違いの比較のためにも利用することができ，その結果を次の節で示す。ただし，このライブラリは新しい版のオペレーティング・システムのリリースにより，Intel Core 2 Duo プロセッサ用にもチューニングされたので，かなり改善されている。

今回は，実数波形のスペクトル解析とその復元であるが，vDSP ライブラリの実数用の高速フーリエ変換の関数を使って実装を行なった。離散コサイン変換にチューニングされたアルゴリズムを使うよりも，一般で出回っている高速フーリエ変換のライブラリで充分な性能を得られることがわかっているからである[20]。入力波形に関して，高速フーリエ変換を適用し，その後逆変換を行なう。何回か行ない，元の波形と一番誤差が小さいものを解析結果として保存し，発振器の方でそれを読み込んで，入力波形に等しいような音声合成を逆高速フーリエ変換で行なう。$2^{12}=4096$個の標本点で行なえば，約11Hz～22000Hzまでの範囲の周波数を合成することができる。これは，人間の可聴周波数の範囲に等しい。このシンセサイザーの機能は一般には，サンプラー（sampler）と呼ばれるものと同じ働きをすることになるが，ここでの面白みは，波形そのものではなくて，離散フーリエ変換されたスペクトルの一連の値をデータとして保存し，そこから波形の合成を行なうことにある。

3．比較・考察

3．1．異なるベクトル演算器を持つ CPU での比較

PowerPC G4の AltiVec[21]ベクトル演算ユニットと Intel Core 2 Duo の持つ Streaming SIMD Extensions（以下 SSE）[22] というベクトル演算ユニットの性能評価になっている。SSE は，Pentium III から始まる開発過程において，SSE/SSE2/SSE3という 3 つの版が用意されている。ただし，Intel Core 2 Duo プロセッサで利用可能なのは，SSE と SSE2までである。AltiVec は，128bit のレジスタを32本搭載したもので，単精度浮動小数点数の場合，128個の要素を持つ配列に対して計算を行なうことができる。また，SSE2では，128bit のレジスタを 8 本搭載したもので，同じく単精度の場合は，32個の要素を持つ配列に対して同時に演算を行なうことができる。

性能評価のために，２つの波形を用意した。１つは，正弦波を，位相を変えて重ね合わせた複素数の波形であり，もう１つは矩形波である。矩形波の場合は，フーリエ解析によって低い周波数から，かなり高い周波数の正弦波に分解される。たとえば，$n=18$すなわち，$N=2^{18}=262144$個の標本点から解析を行なった場合，CD の標本化周波数が44100Hz であるから，６秒程度の音声データに対して解析を行なったことになる。実際に，シンセサイザーで用いているのは，$n=12$すなわち，$N=2^{12}=4096$個の標本点で行なっているので，0.1秒の音声データに対して連続的に解析を行ない，フィルタとして解析を行なっている。

---

[20] 前掲，Wikipedia「離散コサイン変換」の項目を参照（閲覧2009）.

[21] AltiVec Fact Sheet, Freescale semiconductor, 2005.

[22] Intel® 64 and IA-32 Architectures Software Developer's Manual, Version released in 2009.

結果は以下のようなものとなった。n ＝12から n ＝18まで，n を 2 ずつ大きくしていったときの実行速度の結果は，以下のようなものとなった。実行速度は，OS の状態にも依存するので，8 回ほど計測して最高値のものを出している。なお，複素数の実数部分と虚数部分，実数データはすべて単精度である。

表 1　2 つの CPU における実行時間

	PowerPC G4 1.67GHz		Intel Core 2 Duo 2.16GHz	
標本点の数	複素数	実　数	複素数	実　数
$2^{12} = 4096$	153 μ 秒	94 μ 秒	33 μ 秒	18 μ 秒
$2^{14} = 16384$	765 μ 秒	457 μ 秒	171 μ 秒	86 μ 秒
$2^{16} = 65536$	5860 μ 秒	3419 μ 秒	841 μ 秒	450 μ 秒
$2^{18} = 262144$	56582 μ 秒	29512 μ 秒	5443 μ 秒	2357 μ 秒

なお，この評価の実施最中に Mac OS X の版が，10.5 Leopard の後継である10.6 Snow Leopard がリリースされ，参照した文献もリリース前のものとリリース後のものとでは，若干仕様が違っていた。リリース後のライブラリでは，Intel Core 2 Duo の SSE/SSE2 に最適化された形で利用することが可能となっている。評価環境は，PowerPC G4 が Mac OS X 10.4上で，Intel Core 2 Duo は，Intel 用にチューニングされた Mac OS X 10.5上で行なっている。10.6上では評価していないが，更にチューニングされているので，更なる高速化が期待できる。

データを見比べると，複素数に比べて，実数の値の方が，計算が速いことがまず挙げられる。だいたい，どちらのプロセッサにおいても，計算が終わるまで，50%から60%程度の時間で実数値の高速フーリエ変換が行なわれているのがわかる。複素数は実数部と虚数部から構成されることに拠るが，高速フーリエ変換のライブラリは，実数に関して，より高速に変換できることの一つの傍証になるのではないかと思われる。次に，プロセッサ間の比較であるが，異なる版のオペレーティング・システム，周波数，およびコアの個数の違いがあるが，PowerPC G4 の方が遅いのは一目瞭然である。しかし，ここで正規化して比べてみたいと思う。たとえば，2^{14}の標本点数で，実数のデータ値に関して，次のようにクロック数とコア数の比を掛け合わせて正規化してみると次のような値になる。

$$457 \times 1.67/2.16 \times 1/2 \fallingdotseq 176\ \mu 秒$$

これでも Intel Core 2 Duo の86 μ 秒に比べても 2 倍以上の遅さになっているが，健闘はしている。数値計算においては，一昔前のプロセッサが，現在の標準のプロセッサに対して，実行速度を正規化しても，2 倍程度の遅さで留まることの方が難しいのが一般的な状況である。そのような意味で，PowerPC G4 のベクトル演算ユニットは，往時においては，充分強力な機能を保持していたと言えるのではないか。また，ベクトル演算ユニットのサイズについては，Intel Core 2 Duo プロセッサの方が1/4の大きさであるが，ユニットのサイズよりも寧ろコアが 2 つあることや，特にキャッシュなどの効果（L2キャッ

シュが G4 は512KB に対して Duo は 4 MB）が計算速度の性能向上に効いていると推測できる。なお，機材がなかったので，PowerPC G4 の後継である PowerPC G5 上での評価はできなかった。

3．2．離散化データの復元の評価

逆フーリエ変換を行なって，元の値がどれだけ復元されているのか計算した。これは，離散フーリエ変換においては，標本化を行なっているため，本来の周波数のスペクトルには存在しないスペクトルが発生するからである。これをエイリアッシング（aliasing）[23]と呼んでいる。また，一定の周波数を保たないような音声が入力された場合，そのどの部分を標本の対象とするかによって，解析結果が変わってくる。そのため，通常は解析の対象となる波形の一部分を選ぶ窓（一定の時間の幅）を用意して，更に選択された信号に対して，窓関数（window function）を適用することが行なわれている[24]。今回，入力として連続データがくるために，窓を 4 個程度用意し，一旦離散化フーリエ変換を行ない，逆変換をして波形を復元し，元の波形と比べて誤差の平均値（途中は，絶対値を得るために誤差の二乗を求めておき，最後に平方根を取る）が最小になったときのものを解析結果として保存している。ここでは，vDSP ライブラリの誤差の大きさを測定した。以下の表は，先ほどの矩形波（振幅は 1 で，変位は 1 あるいは－ 1 の値を取る）に対して，実数の高速フーリエ変換を行ない，逆変換で復元された波形と元の波形との差の平均値を求めたものである。

表 2　誤差の平均

標本点の数	誤差の平均
$2^{12} = 4096$	0.0156
$2^{14} = 16384$	0.0078
$2^{16} = 65536$	0.0039
$2^{18} = 262144$	0.0020

当然のように，標本点の数が多ければ多いほど，誤差は少なくなってくる。標本点の数の 2 のべき乗数が 2 つ大きくなると，誤差の平均も半分になっていることがわかる。

3．3．波形表示の実装におけるライブラリの評価

CoreAudio のライブラリで図形表示に用いられているのは，Mac OS X 固有のライブラリである Carbon[25]ライブラリまたは，Cocoa[26]ライブラリに拠っているが，今回は

---

(23) はじめて学ぶディジタル・フィルタと高速フーリエ変換，前掲書，pp.147-148.

(24) 同書，pp.149-156.

(25) Carbon Overview, Apple Computer Inc., Version released in 2005.

(26) Cocoa Fundamentals Guide, Apple Computer Inc., Version released in 2006.

Cocoa ライブラリの方は，Objective-C[27]の言語で記述しなければならないので，使用を避け，Carbon ライブラリで表示を行なった。しかしながら，Carbon ライブラリは，C 言語をベースとしているので，オブジェクト指向プログラミングが標準となっているグラフィックス・ライブラリに比べると，かなりプログラミングが煩雑で，オブジェクトごとに手続き（メソッド）が整理されていない印象を持った。特に，2 次元描画ライブラリである Quarts 2D[28]と Carbon ライブラリ上のウィンドウに直接の連携を持たず，HIView[29]というコンポーネントを介さなければならない制約に不満を感じた。今後は，C/C++言語の描画用のライブラリとして，Windows でも共通に使える Qt4[30]のようなライブラリや，Metrowerks 社が Freescale 社に吸収された後にフリーのライブラリとして開発が進められている Open PowerPlant[31]のようなライブラリが，今後 C/C++言語の GUI および描画ライブラリとして利用されることになるのではないかと感じられた。Carbon ライブラリが C 言語ベースで構成されており，使い勝手が悪く，オブジェクト指向を前提とした Cocoa ライブラリが Objective-C という特殊な言語を使わなければならない状況では，後者を前提とした iPhone/iPod Touch といった製品のソフトウェア開発も一部の開発者に留まらざるを得ない。広く開発者を開拓したいのであれば，もっと記述するのが簡単な Python や Ruby などのスクリプト言語から Cocoa ライブラリが利用されるべきで，既にそのような方向で Mac OS X の開発環境が整備されてきている。ただ，Java 言語用の Cocoa ライブラリ[32]が使えなくなったのは残念であり，これもフリーで開発されることを期待したい。また，Windows や Linux 版の Cocoa ライブラリもフリーで開発される方向になることを期待したい。新しい Mac OS X の版（10.6 Snow Leopard）からは，Quartz 2D のライブラリが，Core Graphics に統合されて，すべてのライブラリが再編されている状況である。また，膨大な量の数値計算に対して，Graphic Processor 側が持っているベクトル演算ユニットを利用できる OpenCL[33]というフレームワークも提供されるようになった。これらの新しい環境による性能評価も将来は考えていきたい。

4．結論

正弦波を重ね合わせて音を合成するハモンドオルガンと同じ仕組みの発振器を持つシンセサイザーを実装することができた。また，取得した音源から，離散フーリエ解析によって正弦波の重ね合わせとして周波数スペクトルに分解し，そこから音を合成するシンセサイザーも作成した。また，この離散フーリエ解析の高速化アルゴリズムである FFT のライブラリを利用して，PowerPC G4 と Intel Core 2 Duo という異なる CPU のベクトル

---

(27) Introduction to The Objective-C 2.0 Programming Language, Apple Computer Inc., Version released in 2009.

(28) Quartz 2D Programming Guide, Apple Computer Inc., Version released in 2009.

(29) HIView Programming Guide, Apple Computer Inc., Version released in 2007.

(30) Qt 4.5 White Paper, Nokia Corporation, 2009.

(31) Open Sourced PowerPlant Frameworks, Issac Wankerl, http://sourceforge.net/projects/open-powerplant/, reviewed in 2009.

(32) Introduction to Cocoa Tutorial for Java Programmers, Apple Computer Inc., Last released in 2006.

(33) OpenCL Programming Guide for Mac OS X, Apple Computer Inc., First released in 2009.

演算ユニットの性能評価を行なった。クロック周波数やCPUのコアの数，およびキャッシュのサイズで劣るPowerPC G4でも，サイズの大きいベクトル演算ユニットを持つことから，一昔前のCPUが予期以上の性能を挙げられたことも記録に留めておきたい。

謝　辞

この研究は，本学の個人研究助成費の支援の下で行なわれた。この助成費の申請を評価された委員には感謝の意を伝えたい。また，本学の名誉教授であられる故山本英男先生からは，「情報系の教員は数式の記述のない論文を，論叢や紀要に載せるな」という注意を常に戴いていた。そのため，フーリエ変換を再勉強してこの研究に臨んだ。また，山本先生が亡くなれてから，本学の論文において，数値解析系の論文が絶えて久しい。コンピュータの主要な目的の一つが数値計算・数値解析であるので，情報系の教員からこの論文を上梓することにより，山本先生の遺志を繋げたく思う。さらに，熊田禎宣先生が亡くなられて，学部の理系的なアカデミズムが衰退するのではないかという危惧がある。熊田先生とは研究において直接仕事をさせて戴いたことはなかったが，学部のアカデミズムを保つために奔走されていたことは記録に留めておきたい。3年次までは主に私のテーマ研究会に属した，あるいは主に私の科目を履修しながらも，最後は熊田先生のお陰で卒業論文を上梓し，大学院まで進めた学生が少なからずいた経緯も踏まえ，本稿の最後でお礼申し上げたい。

〔抄　録〕

政策情報学部での表現メディア論の講義の中で，音声について扱う授業の回において，FM 音源の原理を説明するために，以前は一般のシェアウェアのソフトウェア・シンセサイザーを用いていた。このシンセサイザーは倍音となる正弦波を組み合わせて，音を合成していくもので，1970年代のハモンド（ポリフォニック）オルガンと原理を同じにしている。各正弦波については，音圧と位相を変えることができた。ただし，合成される波形がビジュアル的に表現されず，文系の学生の直観に訴えるには，やや乏しいと感じていた。オペレーティング・システムの版が変わり，このソフトウェアは使えなくなったため，これと同機能を持ち，かつ合成波がビジュアル的に表現されるソフトウェア・シンセサイザーを試作することをこの研究の目的とする。および，昨今のシンセサイザーではサンプリング音源に対してのエフェクター機能が備わっている。これと同等な機構を用いて，入力された音声に対して，リアルタイムに高速フーリエ変換を適用し，倍音系列の正弦波に分解する機能を追加する。この分解された正弦波を再び合成したものを音声として出力することにより，正弦波による分解と合成機能，すなわちそれは FM 音源への分解と合成機能がどの程度，実際の音色を再現するに足り得るものであるかを，聴講している学生に直観的に提示することができる。その解析ソフトウェアの試作も行なうものである。