5-4 MSPプログラミングのガイドツアー

いよいよ、これらからMSPによるオーディオ処理プログラミングの内容に入っていこう。ここでは、MSPプログラミングのイントロダクションとして、1つのガイドツアーを企画してみた。できるだけ具体的な素材で、MSPプログラミングの基本を大まかに眺めてみよう。テーマは"シンプルなシンセサイザーを作る"ことだ。この制作を通じて、MSPでデジタル・オーディオがどのように扱われ、MSPプログラミングとはどのようなものなのか、まずはその感触をつかんでほしい。

シンセサイザーの仕組み

よく耳にするシンセサイザーという電子楽器は、直訳すれば合成機になる。何を合成するかと言えば、もちろん音、サウンド、音響である。つまり何もないところからさまざまな音を作り出してしまう音響合成機械なのだ。通常、市販されているシンセサイザーの多くは、鍵盤型のコントローラーと一体化して、それ1台で音響合成と、それを使った演奏まで行えるようになっている。

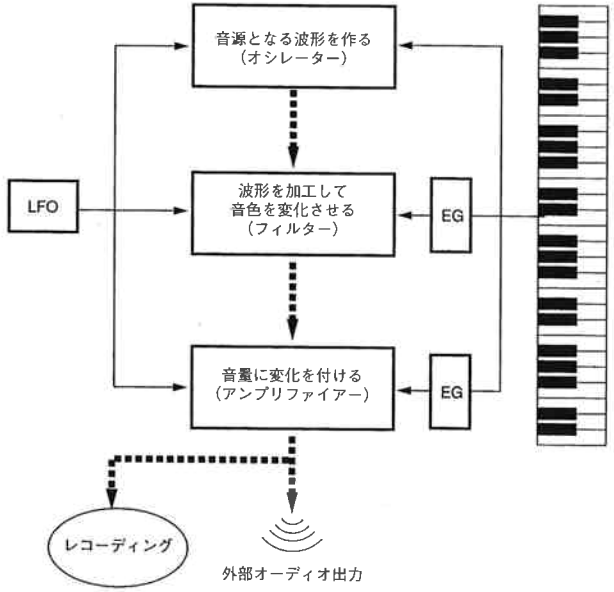
シンセサイザーにはアナログ方式とデジタル方式があり、歴史的にはもちろんアナログ方式の方が先に発達した。現在、シンセサイザーの主流はデジタル方式になっているが、それでも独特のサウンドを求めて、アナログ・シンセサイザーの愛好家も少なくはない。

シンセサイザーの仕組みを単純化すると5-4-1のようになるだろう。オシレーターとは発振器を指し、波形を作り出す源である。これは音響合成の素材となる部分だ。次にフィルターは、オシレーターが作り出す波形を加工し、音色に変化を付ける部分である。一般にはローパス・フィルターと呼ばれるフィルターが用いられることが多く、設定された周波数(カットオフ周波数)より高い周波数成分を通過させないように働く。アンプリファイアーは増幅器のことで音量をコントロールする。この3つの主要部分はシンセサイザーが音を鳴らすための最も中心になる基本的な柱の部分になる。

続いてコントロールに関する部分を見ていこう。右側の鍵盤は手動でシンセサイザーの動作をコントロールするコントローラー部分で、まず弾いた鍵盤の位置によりオシレーターの周波数がコントロールされる。EGはエンベロープ・ジェネレーター(Envelope Generator)の略で、時間とともに変化するエンベロープを作り出す。これはフィルターやアンプリファイアーなどシンセサイザーの各部分につなぐことができ、鍵盤コントローラーを弾くと、自動的にこのエンベロープがフィルターのカットオフ周波数と、アンプリファイアーの音量を時間的にコントロールする。

一方、LFOはLow Frequency Oscillatorの略で、音として聞くためではなく、シンセサイザーの各部分を周期的にコントロールするために用いられる非常に低い周波数のオシレーターのことだ。EGと同様LFOもシンセサイザーの各部分につなぐことができ、LFOがオシレーターの周波数を周期的に変化させるとビブラート効果、アンプリファイアーを周期的に変化させるとトレモロ効果が生まれる。

■5-4-1 シンセサイザーの仕組み

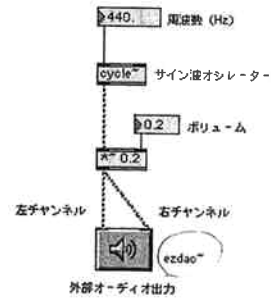


サイン波を出力するcycle~オブジェクト

オシレーターとは、他の力を借りずに、それ自身の力で動作するエンジンのようなものだ。この動作の結果、信号が生み出され、シンセサイザーではそれを音源として利用する。オシレーターには、周期的な波形を繰り返し発生させるタイプと、ランダムなノイズを発生させるタイプがあり、MSPでも多数のオシレーター用オブジェクトが用意されている。ここでは周期的な波形を繰り返し発生させる代表として、サイン波シグナルを出力するcycle~オブジェクトを取り上げよう。サイン波は最も単純な波形として知られており、そのサウンドは基音以外の倍音成分を一切含まない。このことにより特に純音とも呼ばれ、音響合成の中でも重要な波形になっている。

5-4-2のパッチを作って、スピーカー風アイコンのezdac~オブジェクトをクリックすると、電話のツー音のような音が聞こえるはずだ。音は左右のスピーカーの中央から聞こえ、周波数のフロート・ナンバー・ボックスをマウスでドラッグすると、音の高さが上下する。ボリュームのフロート・ナンバー・ボックスを0～1までの範囲でドラッグすると、音の大きさが変わる(注意:くれぐれも耳や再生装置を痛めないよう、1を超えないようにゆっくり慎重にドラッグすること)。

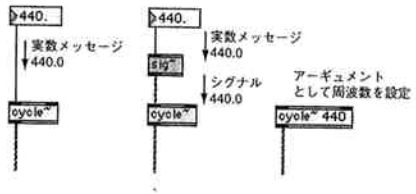
■5-4-2 cycle~オブジェクト(サイン波オシレーター)



cycle~オブジェクトは、第1インレットに実数メッセージもしくはシグナルを受け取ると、それをHz単位の周波数の値として、その周波数で繰り返し－1～1の範囲のシグナル

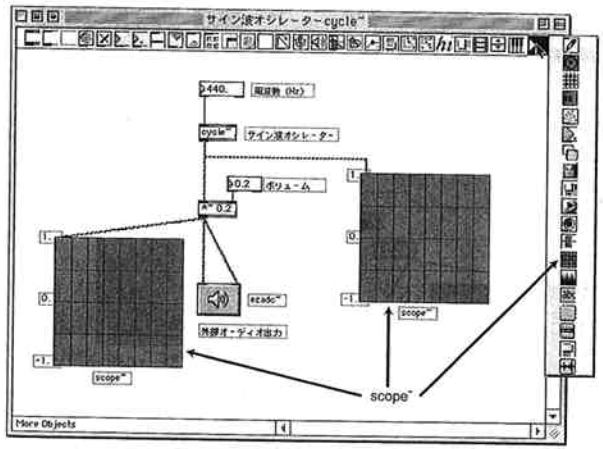
を出力する。このシグナルの時間的変化がサイン波の形状をしている。第1インレットに周波数値を与える変わりにアーギュメントとして周波数の値を書き込んでもよい。第2インレットには位相(0～1)の値を受け取る。

■5-4-3 cycle~オブジェクトの周波数設定方法



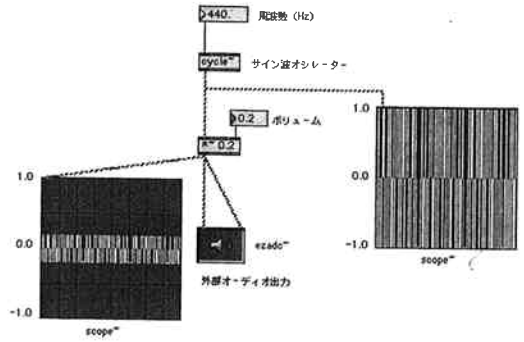
ここで、cycle~オブジェクトが本当にサイン波形のシグナルを出力しているかどうかscope~《Signal Scope》オブジェクトを使って確認してみよう。scope~はオシロスコープ仕様で、受け取ったシグナル値の時間的変化を視覚的に表示させるオブジェクトだ。オブジェクト・パレットからscope~オブジェクトを取り出し、cycle~オブジェクトのアウトレットおよび*~オブジェクトのアウトレットにパッチングする。

■5-4-4 scope~オブジェクトを使う



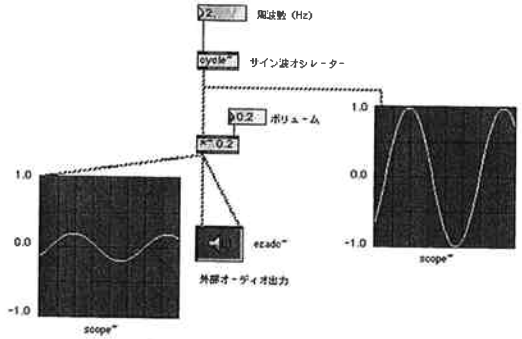
この状態でオーディオ処理をスタートさせると、scope~オブジェクトは受け取ったシグナルの時間的変化(波形の時間領域表現)を表示する。

■5-4-5　scope~オブジェクトでシグナルの時間的変化を見る



しかし、これでは、波形表示が細かすぎてどんな波形なのか確認できないので、cycle~オブジェクトの周波数を極端に小さくして2Hzにしてみる。もちろん人間の可聴周波数の範囲はおよそ20～20,000Hzなので、2Hzの音は聞こえないが、波形をグラフィカルに確認するには手軽な方法になる。

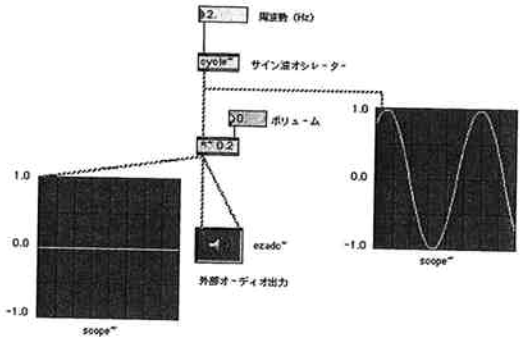
■5-4-6　周波数を小さくして波形を見やすくする



これでcycle~オブジェクトの出力シグナルをグラフィカルに確認することができ、確かにサイン波の形状で繰り返し同じ周期的波形を出力し続けていることが分かる。scope~オブジェクトの縦軸はシグナル値を示しており、初期設定でその表示範囲は−1～1なので、cycle~オブジェクトの出力シグナルは−1～1の範囲だということも分かる。

一般にボリューム(音量)を上げたり下げたりすることは、実際には振幅を大きくしたり小さくしたりすることであり、MSPでは *~オブジェクトでシグナルに乗算を行うことにより実行する。加算や減算ではない点が重要である。−1～1の範囲で変化するシグナルに0.2を乗算すると、その結果、−0.2～0.2の範囲で変化するシグナルになり、ボリュームは5分の1に下がる。乗算する値を0にすれば、scope~オブジェクトの表示は0の値で直線のまま変化しなくなり、ボリュームも0になる。

■5-4-7　ボリュームを0にする



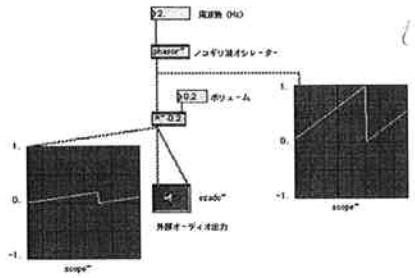
05/3/27

ノコギリ波を出力するphasor~オブジェクト

phasor~は、ノコギリ波を出力するオシレーター・オブジェクトであり、出力シグナルの時間的な変化は鋸歯(ノコギリの歯)状の形をしている。先ほどのcycle~オブジェクトを使ったパッチを使い、周波数を大きくしてノコギリ波のサウンドを聞き、次に周波数を小さくして波形の形を確認してみよう。

scope~オブジェクト表示でも分かる通り、phasor~オブジェクトは0から1へと直線的に変化するシグナルを繰り返し出力している。ノコギリ波のサウンドは後述するように多くの倍音成分を含んでおり、サイン波と並んで音響合成では重要な波形になる。

■5-4-8 phasor~オブジェクト(ノコギリ波オシレーター)

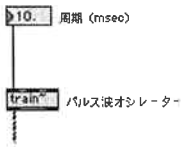


パルス波を出力するtrain~オブジェクト

次にパルス波シグナルを出力するオシレーター・オブジェクトtrain~を紹介する。パルス波は矩形波とも呼ばれ、そのサウンドは奇数倍音(第3、第5、第7……など)を豊富に含む特徴を持っている。train~オブジェクトは、周波数の与え方で、cycle~やphasor~オブジェクトとは異なっているので注意が必要だ。train~オブジェクトの第1インレットには周波数ではなくパルス波の1周期の長さをmsec単位で与える。

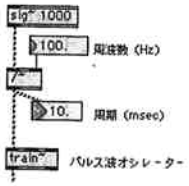
この周期という表現は少し分かりにくいかもしれない。cycle~やphasor~オブジェクトと同じように周波数により音の高さをコントロールする方がなじみやすいようであれば、周波数を周期に変換するようパッチを工夫してみよう。

■5-4-9 train~オブジェクトの周期設定



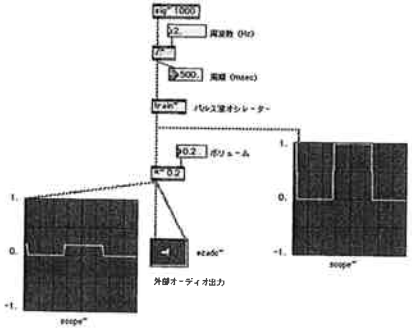
周波数とは1秒(1,000msec)間に繰り返される周期の回数だった。したがって、この周期をTとし、周波数をFとした場合、F=1,000/Tとなる。例えば、周期Tが10msecだとすると、周波数Fは、1,000/10=100Hzになる。逆に周波数Fから周期Tを計算するには、T=1,000/Fとなり、パッチで表現すると5-4-10のようになる。オーディオ処理を開始すると、sig~は1000という値のシグナルを出力し続け、/~は、そのシグナルに対し、第2インレットに受け取った数値(ここでは実数100.0)で除算を行う。こうして1,000/Fが計算され、計算結果としての周期の長さはシグナルとしてtrain~に渡される。

■5-4-10 周波数を周期に変換する



これでcycle~やphasor~オブジェクトと同じように周波数でtrain~オブジェクトをコントロールできるようになったので、周波数を大きくしてパルス波のサウンドを聞き、次に周波数を小さくして波形の形を確認してみよう。

■5-4-11 train~オブジェクトの波形を見る



MIDIノート・ナンバーを周波数へ変換するmtofオブジェクト

MIDIやそれを前提にしたMaxプログラミングに慣れ親しんだ人には、音の高さ(音高)を周波数ではなく、ノート・ナンバーで表す方が分かりやすいかもしれない。MSPによるオーディオ信号処理プログラミングを進めていく上では、周波数の考え方は非常に大切であるが、一方でこれまで親しんだノート・ナンバーの考え方と、周波数がどのような関係にあるのかを理解することも大切である。

まず、MIDIノート・ナンバーについてもう1度整理してみよう。MIDIノート・ナンバーとはMIDIコントローラーの鍵盤に左から右に順に0～127の数字を割り当てたもので、これによって音高を表す。例えば中央ドの鍵盤には60、中央ラの鍵盤には69という数字が割り当てられているが、これに特に理由があるわけでなく、あくまでMIDI規格としての共通の決め事だ。

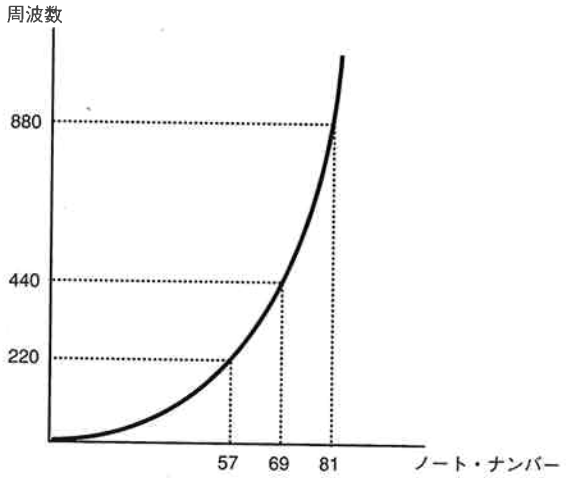
ノート・ナンバーの考え方の背景には、西欧音楽の12音階がある。つまり1オクターブを12個の半音に分け、ノート・ナンバーが1つ増えるごとに、音高は半音ずつ上がるようMIDI機器は規格化されている。その結果、1オクターブ上の音高はノート・ナンバーでは12を足したものになる。例えば中央ラ(69)の1オクターブ上のラは81になる。

■5-4-12　MIDIノート・ナンバーの鍵盤への割り当て

一方、周波数の考え方はこれと大きく異なる。周波数は物理現象としての音を分析して得られたものだ。高さ(ピッチ)を感じられる音は、周期的に繰り返される波形を含んでおり、この周期的な波形が1秒間に何回繰り返されるかという回数が周波数になる。これはMIDI規格のような約束事ではなく、音の物理的な性質を表している。したがってドレミファ……という音名やMIDIノート・ナンバーによって表される音高が、どれくらいの音の高さなのかは、最終的には周波数で定義されることになり、中央ラ(A)の音高は、440Hz(あるいは442Hz)と決められている。正確に楽器を調律し、多数の楽器でアンサンブル演奏を行うためには、こうした基準は欠かせない。

さらに、ノート・ナンバーと周波数の考え方は、音程関係でも大きく違っている。ある音高の1オクターブ上の音高は、ノート・ナンバーでは12を足したものだったが、周波数では2倍になる。逆に1オクターブ下の音高は、ノート・ナンバーでは12を引くが、周波数では2分の1になる。例えば中央ラ(A)の音高は440Hzだが、その1オクターブ上の音高は880Hz、1オクターブ下の音高は220Hzになる。こうしたノート・ナンバーと周波数の関係を図示すると5-4-13のようになる。

■5-4-13　ノート・ナンバーと周波数の関係

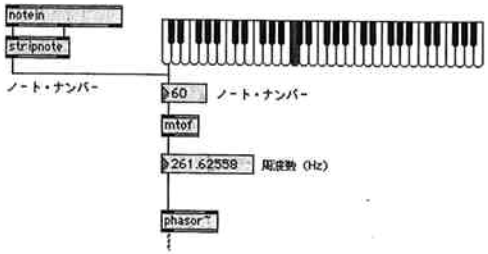


このようにノート・ナンバーは加算や減算といった考え方で操作するが、周波数は乗算や除算によってこれを行う。また音程(2つの音高の違い)はノート・ナンバーでは差、周波数では比の関係になる。

以上の説明はオーディオ信号処理を理解する上で非常に基本的な事項だ。しかし、実際のMSPプログラミングにおいて、例えば中央ドの周波数が何Hzになるかをいちいち計算するのは大変面倒である。このためノート・ナンバーを周波数に変換するオブジェクトとしてmtof、逆に周波数をノート・ナンバーに変換するオブジェクトとしてftomが

用意されている。5-4-14のパッチはmtofオブジェクトを使ってノート・ナンバーを周波数に変換してphasor~オブジェクトに与えている例だ。ノート・ナンバーは外部MIDIコントローラーから入力することもできるし、キーボード・スライダー上をクリックすることで入力することもできる。

■5-4-14　mtofオブジェクトでノート・ナンバーを周波数へ変換

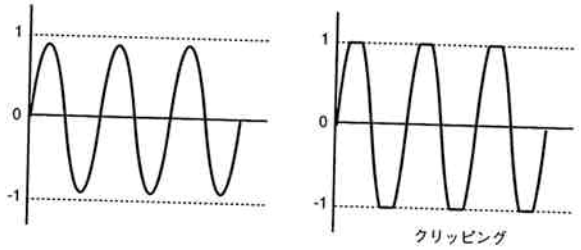


ピーク・レベルを監視するmeter~オブジェクト

先ほど紹介した各オシレーター用オブジェクトを鳴らすパッチでは、ボリューム(音量)を上げたり下げたりすることは、実際には振幅を大きくしたり小さくしたりすることであり、MSPでは*~オブジェクトでシグナルに乗算を行うことによりこれを実行した。MSPでは、外部オーディオ出力するためにdac~やezdac~オブジェクトが受け取る最終的なシグナルは、−1〜1の範囲内になければならない。もしこれを超えた場合、耳障りなクリッピング・ノイズが発生する。このノイズは、場合によっては耳や再生機器をダメージを与える可能性もあるので、ノイズを避けるよう十分注意する必要がある。

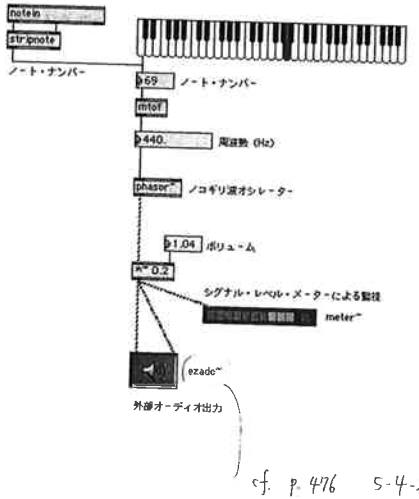
5-4-15は、このクリッピング・ノイズが発生する状況を示している。dac~やezdac~オブジェクトが受け取るシグナルが、−1〜1の範囲を超えた場合、その超えた部分は図のように切り落とされ、1あるいは−1になってしまう。ここでは波形そのものが変化しており、本来の音色とは異なった歪みやノイズが生じてしまう。

■5-4-15　クリッピングの状況



クリッピングを避ける1つの方法として、meter~《Signal Level Meter》オブジェクトによるピーク・レベルの監視がある。このオブジェクトは、受け取ったシグナルのピーク(最大振幅)のレベルを刻々とLEDメーターのように表示し、シグナルの値が−1〜1の範囲を超える場合、赤いLEDが点灯する。ユーザーはこの表示を見ながら、赤いLEDが点灯しないよう適正なボリューム値を設定することができる。ちなみにmeter~オブジェクトではLED1個分がおよそ3dBの音量変化に相当する。またmeter~オブジェクトは、リサイズにより水平メーターにも垂直メーターにもなり、Get Info...により背景、各LEDのカラーなど細かく設定することができる。

■5-4-16　meter~オブジェクトによるピーク・レベルの監視

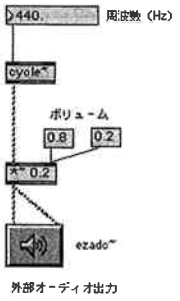


cf. p. 476　5-4-2　"ezdac~"

クリック・ノイズを回避するline~オブジェクト

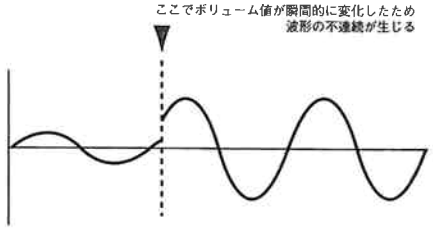
ボリュームをコントロールしていると、操作している瞬間に小さなノイズが聞こえる場合がある。これはクリック・ノイズと呼ばれ、デジタル・オーディオ特有のノイズだ。分かりやすくするために次のようなパッチでクリック・ノイズが発生する状況を確認してみる。

■5-4-17　クリック・ノイズを発生させるパッチ



ここでメッセージ・ボックスをクリックして、ボリューム値を0.8、0.2と交互に切り替えてみると、単に音量が変化するだけでなく、切り替えの瞬間にはっきりとクリック・ノイズが入る。クリック・ノイズが発生する理由は、ボリューム値を瞬間的に切り替えたときに、シグナル値も瞬間的にジャンプし、5-4-18のような波形の不連続が生じるためである。

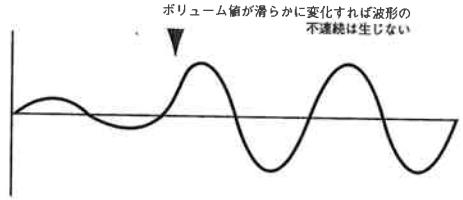
■5-4-18　波形の不連続によるクリック・ノイズの発生



すでに説明した通り、従来のMaxメッセージは、何らかのトリガーが発生した瞬間に弾丸が発射されるようにオブジェクトから出力される。

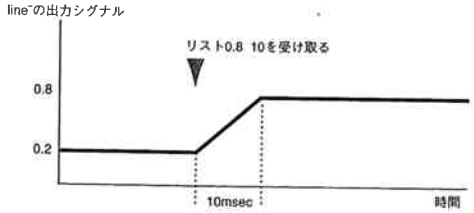
5-4-17のパッチでは、メッセージ・ボックスをクリックした瞬間に0.2や0.8といった実数メッセージが * ~ に渡され、その結果、このような波形の不連続が生じる。オーディオ信号処理では、こうした波形の不連続が起こると大なり小なりクリック・ノイズが発生し、それを回避する工夫がいつも求められる。このノイズを回避するためには、次のようにボリューム値を滑らかに変化させればよい。

■5-4-19　波形の不連続を回避する



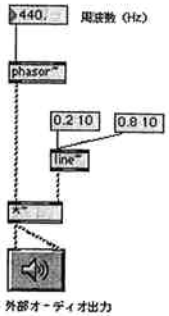
MSPではこのためにline~というオブジェクトが用意されている。line~オブジェクトは、1つの数値を受け取ると、それを単にシグナルに変換するだけだが、2つの数値からなるリストを受け取ると、それを"目標値　目標値への到達時間"と解釈する。例えば0.2 10というリストを受け取ると、現在のシグナル値から10msecかけて0.2へと出力シグナルを連続的に変化させ、0.2に達したらそのままのシグナル値を維持する。次に0.8　10というリストを受け取ると、5-4-20のように10msecかけて0.2から0.8へと出力シグナルを連続的に変化させる。

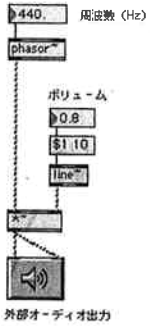
■5-4-20　line~オブジェクトの出力シグナル



このline~オブジェクトを利用して先ほどのパッチを書き換えると5-4-21のようになる。これにより滑らかに変化するシグナルでボリュームをコントロールすることになり、結果としてクリック・ノイズを避けることができる。

■5-4-21　クリック・ノイズを回避するパッチ　■5-4-22　一般的なボリューム・コントロール



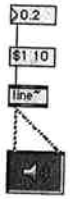


0.2や0.8以外に自由にボリュームをコントロールするためには、メッセージ・ボックスの変数表現を用いて5-4-22のように書き換えるとよい。MSPプログラミングでは、このボリュームだけでなく、周波数やその他さまざまなオブジェクトのパラメーターを滑らかで連続的にコントロールしなければならない場面によく出会うだろう。こうしたとき、line~は非常に有効で重要なオブジェクトになる。

聞こえないシグナル

5-4-22のパッチではphasor~オブジェクトの出力シグナルはオーディオ信号として最終的には外部オーディオ出力され耳に聞こえる音になる。しかし、先に紹介したline~オブジェクトの出力シグナルは、ボリュームをコントロールするためのもので、それ自体聞くためのものではないシグナルだ。試しに次のようなパッチングをしてもまったく音は聞こえない。

■5-4-23　聞こえないシグナルの例



しかし、聞こえるものであれ、聞こえないものであれ、どちらも同じシグナルであることには変わりはない。聞こえるシグナルは極めて短い時間間隔で激しく変化しており、聞こえないシグナルは、変化しないか、あるいはその変化が比較的ゆっくりしているだけの違いなのだ。

重要な違いはシグナルそのものではなく、その用途、目的にある。line~オブジェクトの出力シグナルは、他のMSPオブジェクト（ここでは *~）をコントロールするために用いられている。MSPプログラミングを進めていくと、それ自体は聞くためのものではなく、他のMSPオブジェクトのパラメーターをコントロールするためのシグナルを多数扱うようになるだろう。実際、出来上がったパッチの中では、オーディオ信号として聞くためのシグナルの流れよりも、こうしたコントロール用のシグナルの流れの方が多くを占めている場合も珍しくない。聞こえる、聞こえないに関わらずこうしたシグナルの流れを作り、シグナル・ネットワークを構築することがMSPプログラミングなのである。」

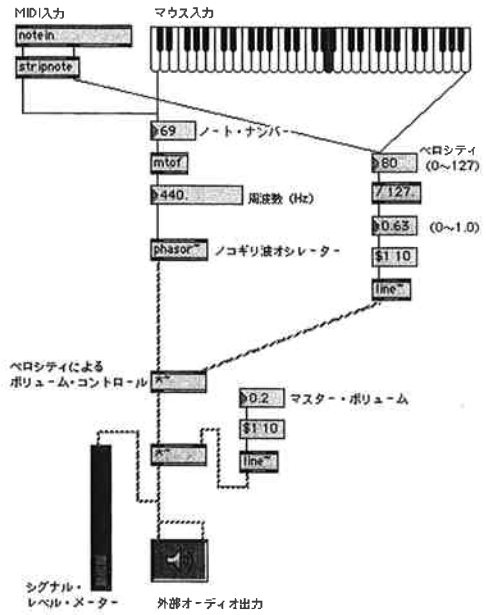
05/3/28

ベロシティにより強弱を付ける（パッチ・オブジェクトの活用）

制作中の簡易シンセサイザーは、外部MIDI入力やkslider《Keyboard Slider》のマウス操作により、オシレーターの周波数をコントロールして、音高（ピッチ）を自由に変えることができる。次にMIDIコントローラーの鍵盤を弾く強さ、あるいはksliderのマウスの上下位置、つまりベロシティによってサウンドに強弱を付けてみよう。

ベロシティといっても何も特別なものではなく、オーディオ処理の上では、ボリュームのコントロールにほかならない。したがって5-4-24のようにパッチを変えればよい。

■5-4-24　ベロシティで強弱を付ける

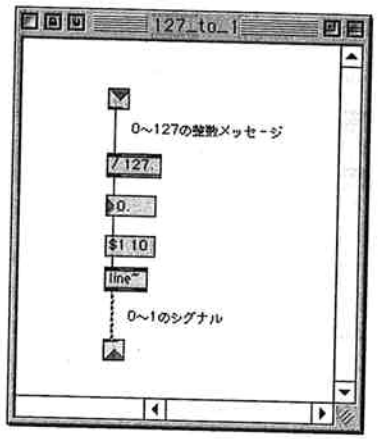


noteinオブジェクトにより受信されたMIDIノート・メッセージはstripnoteオブジェクトによりノート・オフがカットされ、ノート・オンのみ利用される。このベロシティ値がボリュームのコントロールに使われている。ベロシティによるボリュームのコントロールは、先に説明したline~オブジェクトの出力シグナルによる滑らかなマスター・ボリュームのコントロールとほとんど同じだ。ただMIDIのベロシティは0〜127の整数なので、これを0〜1.0の実数に変換し、最終的に0〜1.0のシグナルに変換している。kslideオブジェクトによるマウス入力についても同様である。

この0〜127の整数を0〜1.0のシグナルに変換する処理は、MSPプログラミングの中で頻繁に使われる。なぜなら従来のMaxでは、多くのMIDI関連オブジェクト、またhsliderをはじめとする多くのスライダーやdail《Dial》などのインターフェース系オブジェクトが、0〜127の整数メッセージを出力するように作られているからだ。

したがって、この変換処理をパッチ・オブジェクトとして外部ファイル保存し、さまざまな場面でいつでも使えるようにしておこう。具体的には新規のパッチ・ウィンドウを開き、次のようなパッチを作って保存する。保存先はMaxサーチ・パスの範囲内にする。保存名は127_to_1とでもしておく。

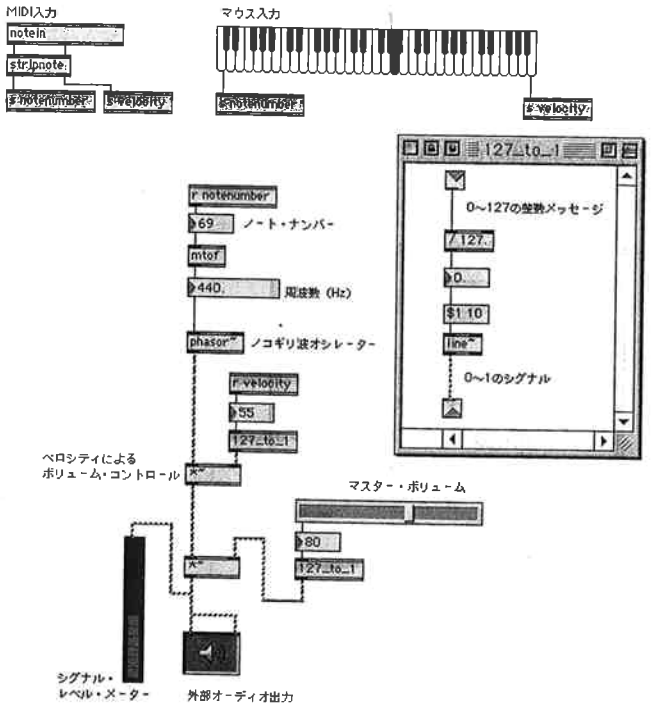
■5-4-25　頻繁に行う処理をパッチ・オブジェクトにする



先ほどのベロシティを付ける5-4-24のパッチを、今保存したパッチ・オブジェクト127_to_1を使って改良したのが5-4-26のパッチである。

5-4-26のパッチでは、パッチ・オブジェクト127_to_1をベロシティによるボリューム・コントロールだけではなく、hsliderと組み合わせてマスター・ボリュームのコントロールにも使っている。またパッチ・コードの配線が少々混乱してきたので、send(s)、receive(r)オブジェクトを使って、ノート・ナンバーとベロシティの数値を受け渡しするように手を加えておいた。

■5-4-26 パッチ・オブジェクト127_to_1の使用

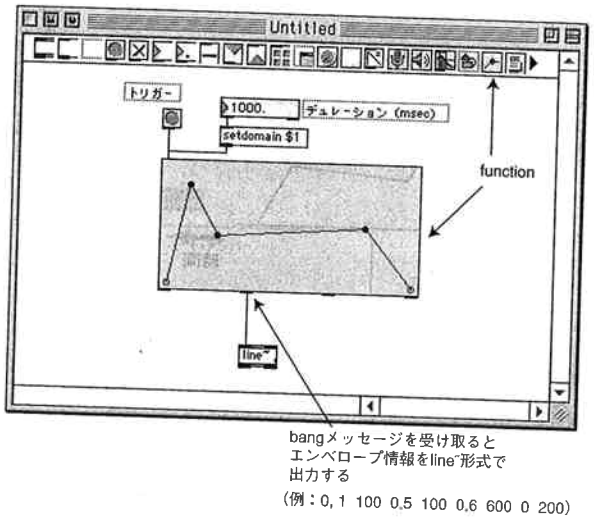


functionオブジェクトとlineオブジェクトでエンベロープを付ける

ある一定時間の波形の振幅の最大値（ピーク）を線で結んでいくと、その音の時間的な音量変化について、おおよその形をつかむことができる。この形をエンベロープと呼ぶ。各種の楽器は、その楽器特有のエンベロープを持っており、例えば打楽器のエンベロープとフルートのエンベロープはまったく異なる。

ここで制作中の簡易シンセサイザーのサウンドにエンベロープを付けてみよう。MSP

■5-4-27 functionオブジェクトによるエンベロープ設定



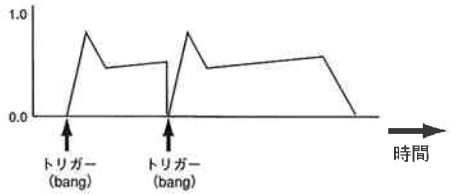
では、function《Breakpoint Function Editor》オブジェクトと先ほどのline~オブジェクトの組み合わせで視覚的に分かりやすいエンベロープ設定が簡単にできる。

パッチ・ウィンドウに置かれたfunctionオブジェクトは、グラフィカルなエンベロープ・エディターとして機能する。エディターの縦軸はエンベロープのレンジを表し、初期設定では0～1になっている。また、横軸は時間間隔（デュレーション）を表現し、初期設定では0～1,000msecだが、"setdomain 実数"というメッセージを与えれば変更することができる。エディター上をクリックすればセグメント・ポイントが追加され、それを自由な位置にドラッグすることで好きな形のエンベロープが描ける。またshiftキーを押しながらセグメント・ポイントをクリックすれば、そのポイントは消去される。

functionオブジェクトは、bangメッセージを受け取るとこうして描かれたエンベロープ情報を第2アウトレットからline~に理解可能な形式で出力する。この形式は2つのメッセージから構成されており、最初に第1セグメントの値が実数で出力され、続いてリスト形式で、第2セグメントの値、それへの到達時間、第3セグメントの値、それへの到達時間……が出力される。だが、ここでの目的は、エンベロープによりボリュームをコントロ

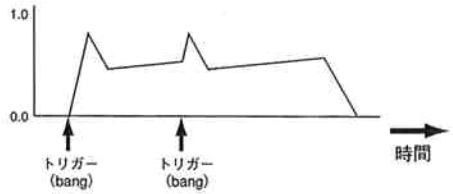
ールすることにあるので次のような状況は避けなければならない。なぜなら瞬間的なボリューム変化により波形の不連続が起こり、クリック・ノイズが発生してしまうからだ。

■5-4-28　エンベロープによる瞬間的な変化



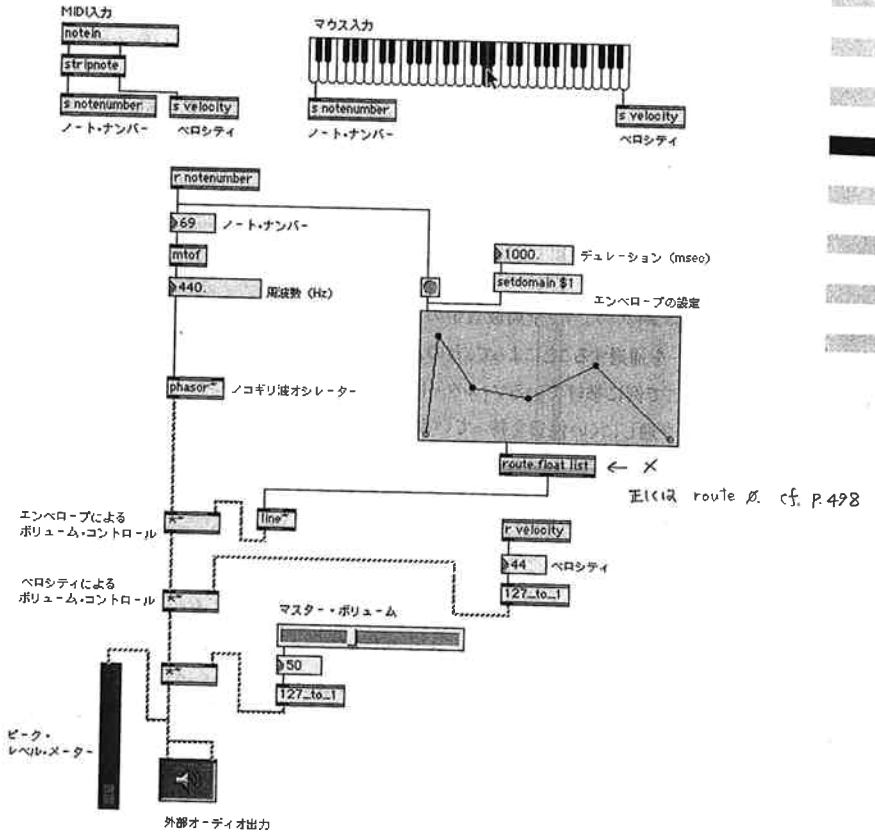
こうした状況を避けるには、エンベロープによるボリューム変化は5-4-29のようにならなければならないだろう。

■5-4-29　エンベロープによる連続的な変化



これを実現するためにはfunctionオブジェクトが最初に第1セグメントの値を実数で出力するのをカットすればよい。5-4-30のパッチでは、MIDI入力あるいはksliderオブジェクトによるマウス入力をトリガーにして、phasor~オブジェクトの出力シグナルにエンベロープを付けているが、functionオブジェクトの出力をrouteオブジェクトで実数(float)とリスト(list)にルート分けし、リストのみline~オブジェクトに送っている。これで設定したエンベロープの形状に従って、発音を始め次第に音が消えていくようになる。ただ音が聞こえなくなっても、それはボリュームが0になっているだけで、オーディオ処理はezdac~オブジェクトをクリックして停止するまで続いている。

■5-4-30　エンベロープを実装する



❺ filtergraph~オブジェクトとbiquad~オブジェクトで音色を変化させる

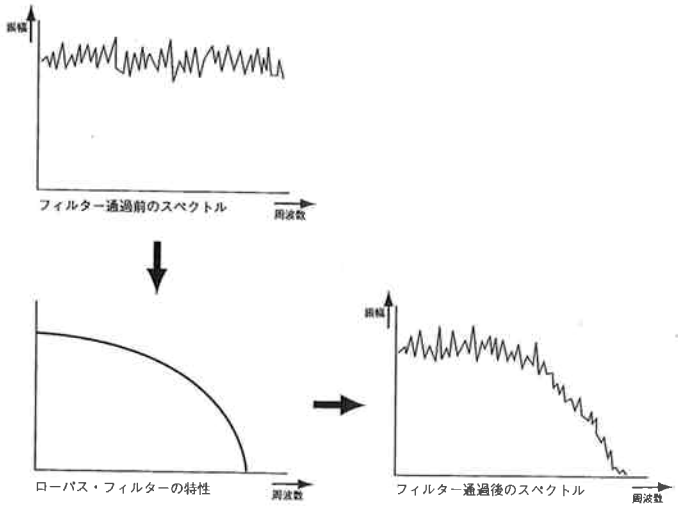
シンセサイザーの心臓部は、やはりさまざまなサウンドを作り出す音響合成部だろう。これまで紹介してきたcycle~、phasor~、train~といったオシレーター用オブジェクトは

それぞれ特徴的な音色を持っており、それらを取り替えることで異なった音色を手にすることができる。しかし、オシレーターをそのまま鳴らす、あるいは取り替えることは、音響合成とはいわない。またこの方法では自由にさまざまなサウンドを作り出すことも望めない。音響合成については「5-5音響合成：加算合成」(P504)以降で詳しく紹介していくが、ここではその第1歩としてフィルターによる音色の変化を試してみよう。これはアナログ・シンセサイザーでもよく知られた方法であり、音響合成では減算合成といわれるテクニックの初歩的なものに当たる。

通常、フィルターと言えば、空気清浄フィルターのように、あるものは通し、あるものは通さない性質を持ったものを指す。オーディオ信号処理では、この通したり通さなかったりする対象は、サウンドに含まれるさまざまな周波数帯の成分になる。5-4-31は、ホワイト・ノイズのような全周波数帯の成分をランダムに含むサウンドのスペクトルが、フィルターを通過することによって、どのように変化するかを示している。

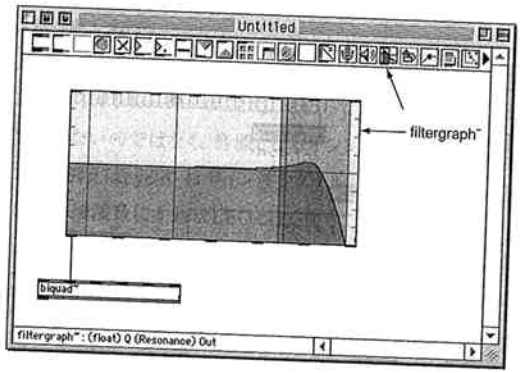
ここで例に挙げているフィルターは、ローパス・フィルターと呼ばれ、高い周波数帯の成分を通しにくい性質を持っている。この結果、元のノイズは高い周波数帯の成分をカットされて柔らかいサウンドになる。

■5-4-31 ローパス・フィルターの機能



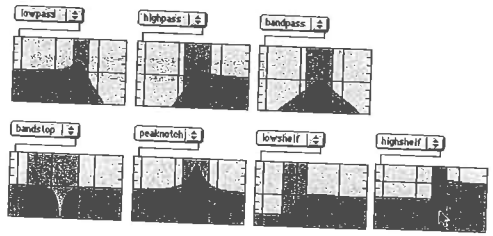
MSPではローパス・フィルターをはじめ、さまざまな特性を持つデジタル・フィルターを個別のオブジェクトとして多数用意しているが、その中でもfiltergraph~《Filter Graph》とbiquad~オブジェクトの組み合わせは、視覚的にフィルター特性をコントロールでき、しかも1つで何役もこなす万能フィルターとして非常に便利だ。

■5-4-32 filtergraph~オブジェクトとbiquad~オブジェクトの組み合わせ



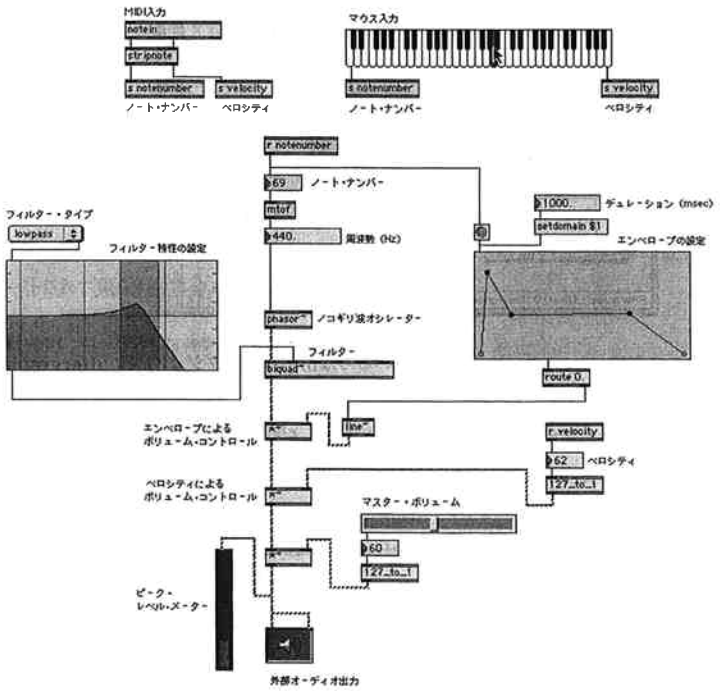
ここで実際にフィルター処理を行っているのはbiquad~オブジェクトで、filtergraph~オブジェクトはそのエディターとなるインターフェースである。マウスによりフィルター特性の形状をコントロールすることができるだけでなく、5-4-33のようにフィルター・タイプの名称をメッセージで与えることにより、タイプを切り替えて、多種類のフィルター機能を使うことができる。フィルター・タイプの名称を書き込んだumenuオブジェクトは、filtergraph~オブジェクトのヘルプ・パッチからコピーすればよい。

■5-4-33 biquad~、filtergraph~で可能になるさまざまなフィルター



制作中の簡易シンセサイザー・パッチにbiquad~、filtergraph~オブジェクトを追加したのが5-4-34のパッチだ。ここでは、オシレーターにphasor~オブジェクトを使っており、phasor~オブジェクトの出力シグナルであるノコギリ波は豊富な高次倍音成分を含んでいるため、フィルター処理を行うことで、さまざまな音色を作り出すことができる。

■5-4-34　フィルターの実装



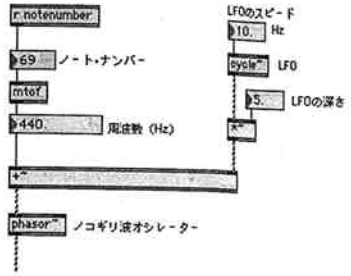
LFOによるパラメーター・コントロール

この簡易シンセサイザー・パッチでは、すべてのコントロールを手動で行っている。MIDI入力、kslideのマウス入力、そしてさまざまなパラメーターのマウスによる設定、すべて手動である。多くのシンセサイザーでは、こうした手動の操作、設定以外に機械的な自動コントロールも可能になっている。その代表がLFOによる周期的なパラメーター・コントロールであろう。

LFOとは、Low Frequency Oscillatorの略称で、可聴範囲を超えた非常に低い周波数で動作するオシレーターのことだ。といっても特別の種類のオシレーターを用意しなければならないのではなく、音源となるオシレーターの周波数をただ小さくしてやるだけでよい。例えばcycle~ はサイン波シグナルを出力するオシレーターだったが、これを非常に低い周波数にすればLFOとして使うことができる。

例えば、5-4-35のようなパッチングでは、phasor~オブジェクトの周波数をcycle~オブジェクトによってゆっくりと周期的に上下させることになり、結果としてビブラート効果が生まれる。LFOとして用いたcycle~オブジェクトは、−1〜1の範囲で1秒間に10回周期的に変化するシグナルを出力し、これが * ~オブジェクトによって−5〜5の範囲の変化となり、440.0と加算されることで、最終的にphasor~オブジェクトの周波数は、435〜445Hzの範囲で周期的に揺れることになる。

■5-4-35　LFOとしてのcycle~オブジェクトの活用

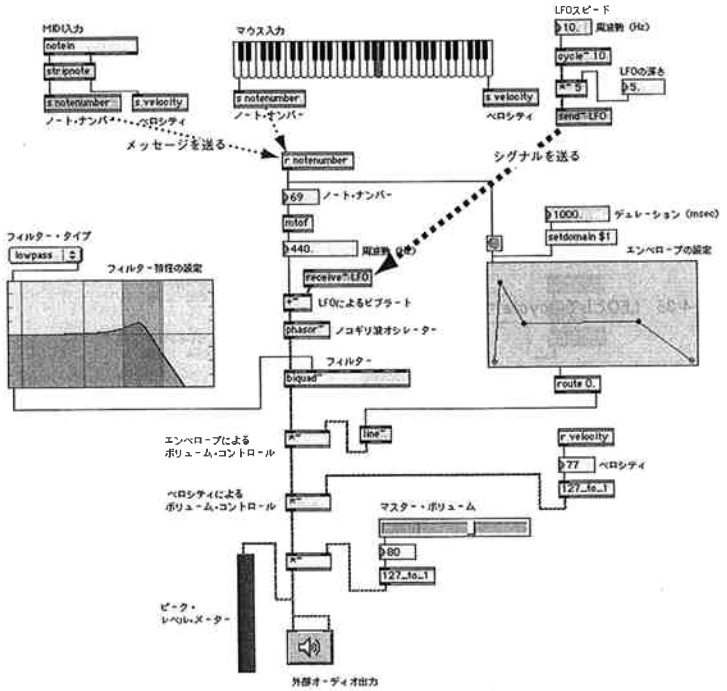


こうしたLFOは、音源となるオシレーターの周波数をコントロールするだけでなく、音量やフィルター特性などさまざまなパラメーターに取り付けることができる。

6-4-34のパッチにLFOによるコントロールを追加したのが次の4-5-36のパッチだ。パッチ・コードが混乱しそうなのでセンド・レシーブを使っている。

ここで重要なのは、センド・レシーブする対象がシグナルの場合、send~とreceive~オブジェクトを使わなければならないことだ。これはsend(s)とreceive(r)のシグナル用オブジェクトであり、アーギュメントとして書き込まれた名前が同じもの同士で、パッチ・コードなしでシグナルの受け渡しを行う。

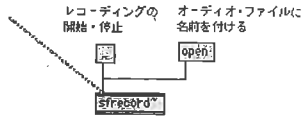
■5-4-36 send~オブジェクトとreceive~オブジェクトの使用



レコーディングとファイル保存

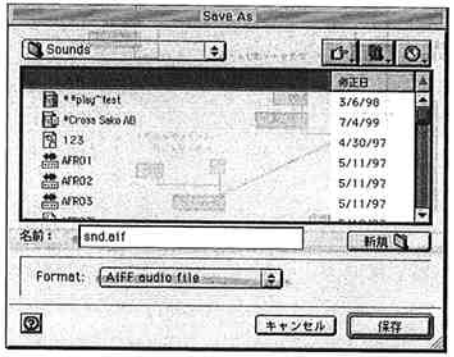
最後に、この簡易シンセサイザー・パッチから生み出されるサウンドをレコーディングして保存する方法を紹介する。ここではsfrecord~オブジェクトによるハード・ディスク・レコーディングを行う。sfrecord~オブジェクトは、コンピューターのハード・ディスクにオーディオ・ファイルを作り、そこに受け取ったシグナルを直接書き込んで行く。次のようなパッチングで簡単に使うことができる。

■5-4-37 sfrecord~オブジェクトによるハード・ディスク・レコーディング



まずopenメッセージを与えると一般的なSave Asダイアログが開き、そこでオーディオ・ファイルの保存名と形式、保存先を指定する。利用できるファイル形式はAIFF、WAVE、au、rawである。

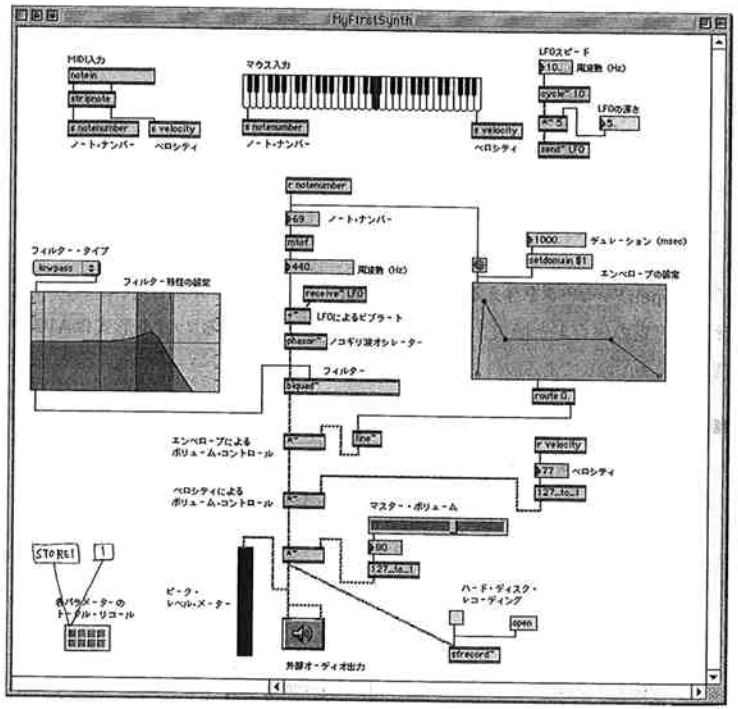
■5-4-38 Save Asダイアログで保存名などを指定



あとは、toggleオブジェクトなどにより1を与えればレコーディングを開始し、0を与えれば停止する。例では、入力インレットが1つでモノラル録音になっているが、アーギュメントでチャンネル数を指定してやれば、ステレオ録音をはじめ最大16チャンネルのマルチチャンネル・レコーディングが可能である。

4-5-39は、簡易シンセサイザー・パッチにsfrecord~オブジェクトによるレコーディング機能を追加したものだ。各パラメーターの設定を一括してパッチに記憶させ、瞬時に呼び出すトータル・リコールのために、presetオブジェクトも追加している。

■5-4-39　簡易シンセサイザー・パッチの完成



05/4/1

次のステップへ

Max/MSPで行うオーディオ信号処理プログラミングとはどんなものかを紹介する目的でこれまで作ってきた簡易シンセサイザー・パッチは、以上のようなものだ。実際にMIDIコントローラーやksliderをマウスで操作して音を鳴らしてみよう。また、さまざまなパラメーター設定を変えて、それがそのような効果をもたらすか実際に体験してほしい。

市販のデジタル・シンセサイザーを使った経験のある人は、得られるサウンドに大いに物足りなさを感じるはずだし、機能的にもっと多くのことを望むに違いない。まさにこうした不満が新しいアイディアを生み、オリジナルのオーディオ処理プログラミングへの道を拓いていくのである。そしてMSPはこうしたアイディアに応えることができるだろう。

例として挙げた簡易シンセサイザーは、これから詳細に解説する音響合成、サンプリング＆プレイバック、エフェクト処理、空間処理、ミキシング処理などの導入であり、発展の糸口になるものだ。どのような処理であれ、それらはすべてデジタル・オーディオ信号処理であり、共通した考え方、方法、テクニックで実現することができる。この簡易シンセサイザー・パッチには、そのエッセンスがたっぷりと詰まっており、筆者が伝えようとしたものその点である。

05/3/29