無限音階の仕組みと作り方

SET/K07 文月師走

1. はじめに

無限音階というものを知っていますか？

音の錯覚の一つで、音高(音程。音の高さ)が無限に上がり続ける、または下がり続ける音のことです。

絶対音感があればそう聞こえないような気もしますが…。

ここでは、その仕組みの一例とその作り方を紹介してみたいと思います。

2. 準備

後で実際に作るのでそれに必要なソフトウェア

- 効果音工ディタ_D：

  http://www.geocities.jp/hirogamesoft/

  効果音工ディタ。無限音階(偽)で使います。

- Audacity：

  http://audacity.sourceforge.net/

  修正に使います。Raw 形式の読み取りもできるので、二つ目でも使います。

記事を書くのに使ったソフトウェア

- NY Spectrum Analyzer：

  http://park1.wakwak.com/~y-nagano/Programs/Spectrum/

  軽くてみやすいので。vector に有るほうはかなり古いです。

  公式に行って初めて分かることですが。(泣)

- WaveSpectra：

  http://www.ne.jp/asahi/fa/efu/soft/ws/ws.html

  上のものよりしっかりしたスペアナ。

  色々オプションが有ります。

- WaveGene：

  http://www.ne.jp/asahi/fa/efu/soft/wg/wg.html

  WaveSpectra と同じ作者の作った信号発生ソフトです。

- MIDKEY：

  http://plaza2.mbn.or.jp/~yoshio2/

  ソフトウェア MIDI キーボード。手元にこれしかなかったので。

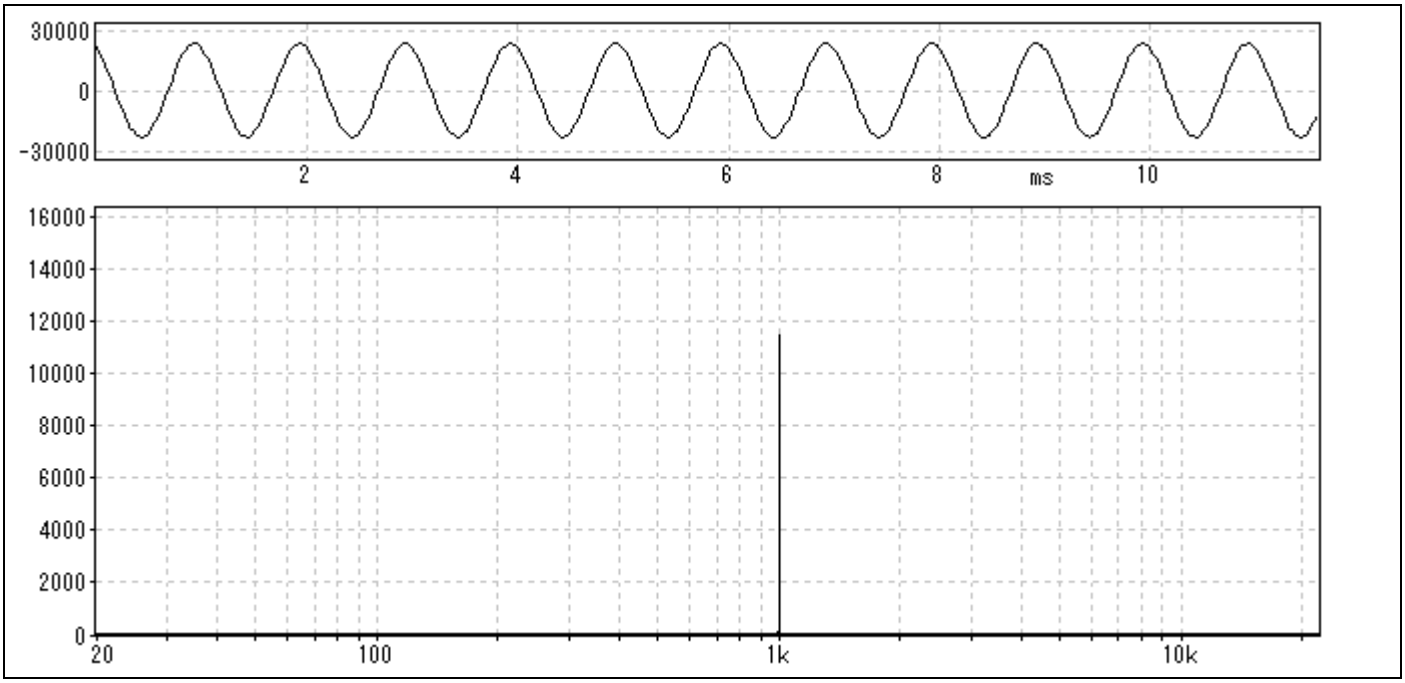
3. まず、音高とは何か。

> 音高(おんこう)、ピッチとは、主に音楽で用いられる言葉で、知覚される音の高さ、もしくは音の物理的な高さ(基本周波数[Hz])のこと。「音高」の聴覚上の概要と物理的な意味(波数)は必ずしも一致しない。音高の呼び名に音名がありアルファベットと数字の組み合わせで表すことが多い。物理的な測定によってある音の基本周波数が決定されたとしても、倍音や部分音(Partial)のために、知覚される音高とは異なる場合がある。人間の聴覚システムは、ある特定の状況下においては、音と音の周波数差を区別できない可能性もある。

出展：Wikipedia

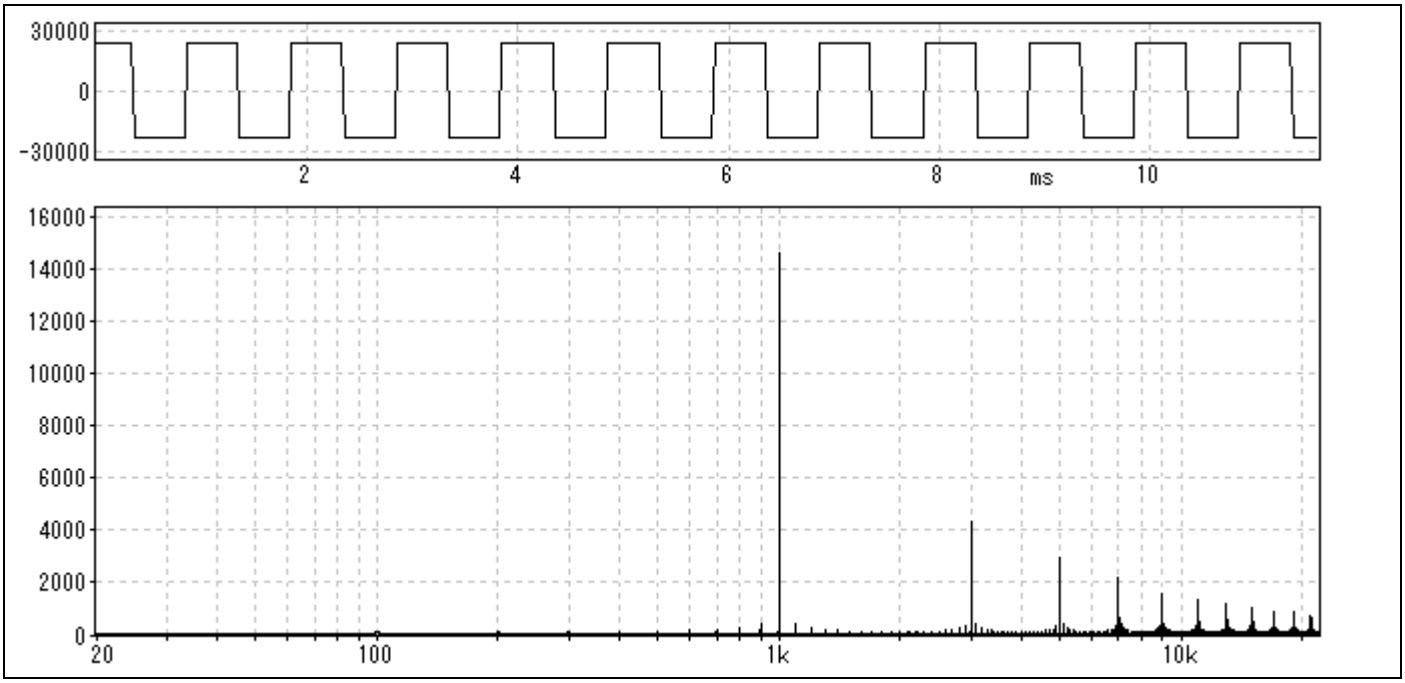
大抵は基本周波数を持ち、その整数倍の周波数を持つ倍音を持ちます。

正弦波の場合。



サイン波とも言います。スペアナではこれ元に調べるので基本周波数しかありません。

矩形波の場合。



さて本題のスペクトルです。

1k の場所が一番強く、次に 3k、さらに次に 5k の部分…と規則的に並んでいるのが分かります。そしてずっと続きます。

これには規則性があって次の数式で表現できます、が。

$$x_{\mathrm{square}}(t) = \frac{4}{\pi} \sum_{k=1}^{\infty} \frac{\sin\left((2k-1)2\pi f t\right)}{(2k-1)}$$

本題に関係ないので置いておきます。

さてこれらの意味ですが、1kHz が基本周波数で 3kHz やその上はその倍音に当ります。

そして、人はこの基本周波数を感じ取ってそれが音の音高だと理解します。

それ以外の部分は音色として解釈します。

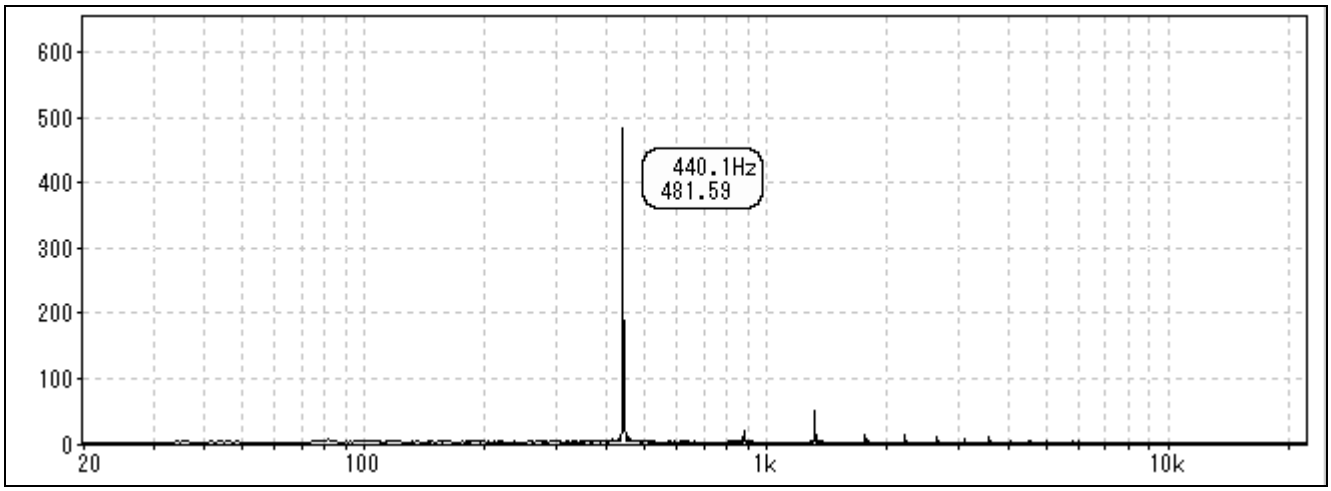
要するに分かりやすい部分だけとってきて、他の部分はその分かりやすい部分の持つおまけとして解釈するわけです。

じっくり聞いていれば複数の音の成分が混ざっているように感じることも出来ますが・・・置いておきます。

じゃあ和音はどうなんだとか言うと、これを複数同時に感じ取ったりしてるわけです。

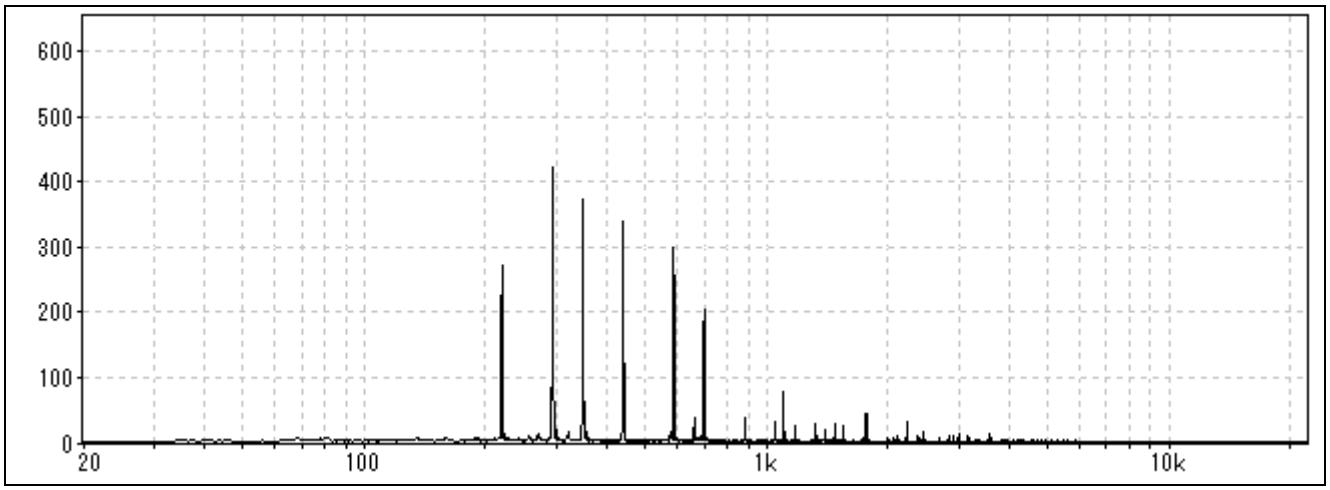
このそういう音はコンピュータで上手く解析するのがとても難しいです。

ためしに



MIDI キーボードを使って出した音です。

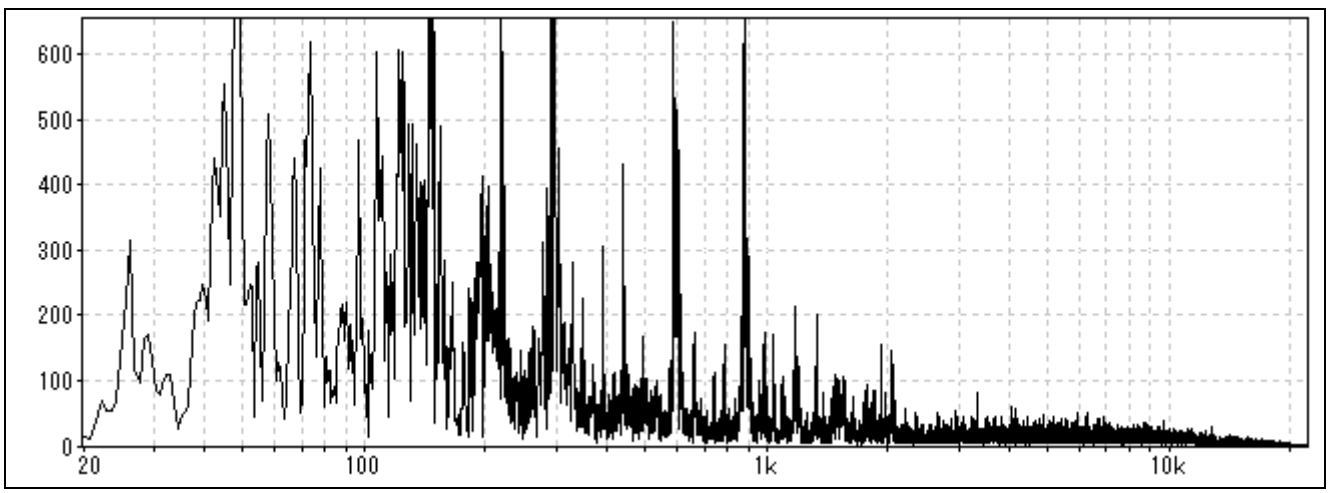
これなら「ラ(A4)、440Hz」が有ると分かりますが…



適当に三つ押してみた音です。Dm ってコードらしいです。

が下のラの倍音が大きくて 4 つ鳴っているようにも見えます。

こうなってくると機械的に調べるのが難しくなってきて…



もう訳が分かりません。

コラム：スペアナ

このように音等の波を解析する道具をスペクトラムアナライザ、略してスペアナと呼びます。

波には「全ての繰り返しを行う波は正弦波の重ねあわせで再現できる」特性があります。何故かは…フーリエ解析とかの講義を受けに行けば分かるのでは無いでしょうか。

スペアナこの原理を元に波を分解する道具です。今回のものでは、横が周波数軸で縦が音量軸となっています。

1kHz の所が 1000 を示す場合、それは 1kHz の正弦波を 1000 含んでいる事を示します。

ちなみに全ての成分をまた合成しなおせば理論上は元の波形が得られます。

補足：

多くの楽器では倍音が強く出ますが、倍音以外も出ます。そういう音も含めて上音と言うらしいです。

4. 作ってみる。無限音階(偽)

説明不足のようにも見えますが、

「基本の音が音高を示し、他の音はそれの音色として解釈される」

事さえ知っておけば、とりあえずそれっぽいものを作るには十分です。

勘がよければもう分かりますが、基本の音がわかり難い音を出して以前出した音に戻ったときに違和感がなくなるようにすればいいだけです。

というわけで作り方…と言いたい所ですが、ブレーカーが飛んで中間データを喪失したので概要と出力のみにします。

作成には冒頭で紹介した効果音工ディタ_D を使います。

Step1

まず基本波形を作ります。

編集->倍音作成で適当に、出来れば単純な正弦波を作ります。

Step2

次に周波数変化を書き入れます。

直線ツールで角から角へ斜線を入れます。

向きはどちらでも構いません。

Step3

音量変化を書き入れます。

山形に入れておきます。

ここの出来が自然な無限音階にする鍵でもあります。

Step4

細かい数値を適当に変えます。

最低周波数が若干低いので上げたりします。

時間も 500ms では短いのでこれも 10 秒ほどに伸ばします。

後で散々音を重ねるので、音割れ禁止もチェックします。

再生位置を 0-300 程度に変更します。

Step5

作ったトラックを他全てにコピーします。

コピーしたトラックを有効にするのを忘れないでください。

コピーするごとに、再生位置を 25～50 程後ろにずらしていきます。

ずらす幅は全て一定にします。

上手く倍音の位置にできればいいですが、ほぼ無理なので気にしません。

Step6

聞いてみて、違和感があれば関連する部分を作り直します。

Step7

Wave で出力し、Audacity で取り込みます。

先端と後端で音が足りていない部分をカットします。

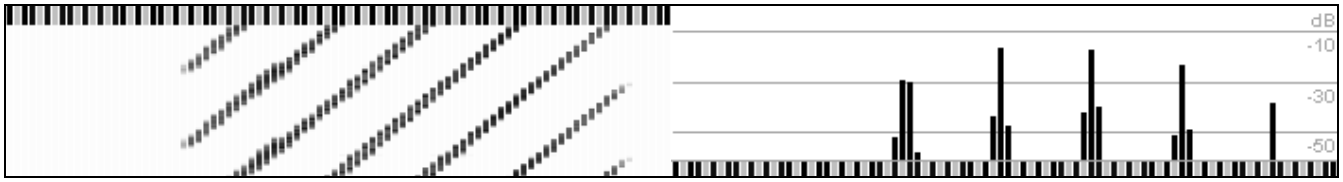
Step8

全体を後ろにペーストして、上手く繋がるよう前半終端を削ります。

出来たら希望する長さになるまでひたすら繋いで終わりです。

この方法で以前作った無限音階もどきを、貼れないので画像で紹介します。

NY Spectrum Analyzer で再生中のスペクトルを時系列で画像化してあります。



この例だと音の数が少ないですが、これをみれば無限音階がどういうものか分かるかと思います。

基本になる音が段々小さくなっていって別の音が気づかないうちに基本の音になってしまうわけです。

この例は音が離れすぎている上に音量の操作が下手なので、基本の音が入れ替わる瞬間が若干知覚しやすくなってしまっています。

そして無段階で変化していて音階になって無いという落ちも付いてしまいます。

この辺りがもどきと書いた理由です…。

5. 作ってみる。無限音階(真)

というわけで、もっとしっかりとした無限音階を造ってみたいと思います。

これ以上の細かい操作は既存エディタでは出来なくは無くとも面倒なので、プログラムを書きます。

```
001  #include <stdio.h>
002  #include <stdlib.h>
003  #include <math.h>
004  #define countof(ary) (sizeof(ary)/sizeof(*ary))
005
006  void help(char **argv){//使用法
007      printf("%s output length base tone span upwaves octave¥n",argv[0]);
008      printf("  output  出力ファイルです¥n");
009      printf("          形式：Signed 16bit PCM Mono 44100Hz Raw Wave¥n");
010      printf("  length  全体の長さ（単位:0.01秒)¥n");
011      printf("  base    基本周波数（単位:0.01秒）デフォルト:440¥n");
012      printf("  tone    1音の長さ（単位:0.01秒）デフォルト: 30¥n");
013      printf("  span    音と音の間隔(単位:0.01秒）デフォルト: 20¥n");
014      printf("  upwaves 上音の数                  デフォルト:  3¥n");
015      printf("  octave  1オクターブの分割数       デフォルト: 12¥n");
016  }
017  const double PI=3.14159265358979323846264338327 95;
018  int main(int argc,char **argv){
019      long l;
020      //パラメータ用数値デフォルト
021      long length
022          ,base   =440
023          ,tone   = 30
024          ,span   = 20
025          ,upwaves=  3
026          ,octave = 12;
027      //入力
028      long *inputs[]={NULL,NULL,&length,&base,&tone,&span,&upwaves,&octave};
029      if(argc<=2){
```

```
030         help(argv);
031         return 1;
032     }
033     for(l=2;(l<argc)&&(l<countof(inputs));l++){
034         *inputs[l]=atol(argv[l]);
035     }
036     //計算用中間数値
037     long subrate=441;               //長さ計算用基本周波数
038     long rate   =subrate*100;       //基本周波数
039     long samples=subrate*length;    //総サンプル数
040     long blank  =subrate*span;      //1間隔無音サンプル数
041     long play   =subrate*tone;      //1間隔発音サンプル数
042     long onekey =blank+play;        //1間隔サンプル数
043     long keycnt =upwaves+1;         //計算上での上音数
044     long display=samples/10;        //経過出力単位
045     long sample;                    //対象サンプル番号
046     FILE *fp;                       //出力ファイル
047     double wavgap=1.0*octave/keycnt;//上音周波数間隔
048     //定番のファイルオープン
049     fp=fopen(argv[1],"wb");
050     if(!fp){
051         printf("outputfile open failure.¥n¥tfilename:%s¥n",argv[1]);
052         help(argv);
053         return 1;
054     }
055     printf("start... ");//状況出力
056     for(sample=0;sample<=samples;sample++){
057         if((sample%display)==0){//状況出力
058             printf(" %02d0%%",sample/display);
059         }
060         //1間隔中でのサンプル数
061         long keypos=sample%onekey;
062         //何間隔目
063         long keynum=(sample/onekey)%keycnt;
064         short dat;
065         if(keypos<play){//発音中
066             double  clock,          //各音に頭合わせしたHzの基本振動
067                     predat,         //中間形式
068                     tone,           //現在の音の周波数
069                     level,rlevel;   //音量
070             tone=base*pow(2.0,1.0*keynum/octave);   //基本音程を求める
071             clock=(2*PI)*keypos/rate;               //位相を求める
```

```
072             level=(keynum+1.0)/keycnt;              //最低周波数の音量
073             rlevel=1-level;                         //最高周波数の音量
074             //人間の音量に対する感覚は指数だか対数だかなんだってサ
075             level*=level;rlevel*=rlevel;
076             //最低周波数
077             predat=(sin(clock*(tone)*pow(2.0,(1.0)/wavgap)))*level;
078             //中間音
079             for(l=0;l<upwaves;l++){
080                 predat+=sin(clock*(tone*pow(2.0,(l+2.0)/wavgap)));
081             }
082             //最高周波数
083             predat+=(sin(clock*(tone*pow(2.0,(upwaves+2.0)/wavgap))))*rlevel;
084             //ノイズ除去。各音の前後1/10をフェードする。
085             if((keypos)<(play/10)){
086                 predat/=(play/10);
087                 predat*=keypos;
088             }else if((play-keypos)<(play/10)){
089                 predat/=(play/10);
090                 predat*=(play-keypos);
091             }
092             //出力用に数値化
093             dat=(short)((predat/(upwaves+2))*0x7FFF);
094         }else{//無音
095             dat=0;
096         }
097         fwrite(&dat,2,1,fp);//1サンプル出力
098     }
099     printf(" end.¥n");//状況出力
100     fclose(fp);
101     return 0;
102 }
```

WAVE 形式で出力すべきかも知れませんが、WAVE 形式を出力するのも面倒なので直接出力してしまいます。

何も最適化していないので動作は恐ろしく低速です。10 分とか出力すると死ぬかも知れませんのでおやめください。

聞くときや編集するときは Audacity で取り込みます。

ファイル→インポート→Raw～で、ファイル形式をあわせて取り込めば OK です。

リトルエンディアン Signed 16bit の PCM でモノラル、サンプリング周波数は 44100Hz となります。

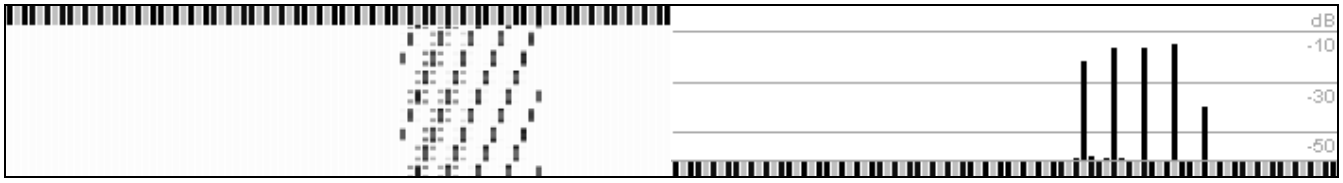
ソースコードを読んでもらえば分かりますが、1 サンプルごとに波形を合成して出力しています。

pow がそこかしこに出てきますが、これは音階が指数だか対数だかにそって並んでいるためです。

1 オクターブで周波数二倍なので「pow(2,位置/分割数)」とすることで対数スケール上にて等間隔に並びます。

上音の計算でも使っていますが、これは倍音やそれに近い効果を期待しているためです。

面白くない解説ばかりしてもつまらないので、実際に聞いてみてください…と言いたいのですが、やっぱりここには貼れないので画像で紹介します。



割としっかり音階になっています。聞いた感じでも切り替わった瞬間がわかり難いので、おおむね成功だと言ってよいと思います。

6. まとめ

いかがでしたでしょうか。

思いのほか単純に実現できるので拍子抜けかも知れませんが、一応これでも無限音階です。

他にも人間の錯覚は色々有るので、試してみると面白いかと思います。

今回は端の音を音量調節で消しましたが可聴範囲外まで持っていくという方法も有りですし、プログラムで作った側もフェードアウトを両端 1 音ではなく複数音操作することでより滑らかに変化させられます。

改善点は山ほどありますが、楽しんでもらえたなら幸いです。