メモ

桂田 祐史

2008年1月23日

卒研のWWWページ http://www.math.meiji.ac.jp/~mk/labo/2007/

1 WAVE ファイルを読む

テスト用の楽器の WAVE ファイルは、例えば『楽器の音の WAVE ファイル』[1] で入手して下さい。

WAVE ファイルの内部構造を詳しく知る必要はないと思うが、参考までに WAVE ファイルを読み取るプログラム readwave.c[2] を紹介しておく。これは WAVE ファイルを読んで、内容をテキスト・ファイルにするプログラムである。

コンパイルは簡単、実行も簡単

```
oyabun% ls
Makefile      piano.wav     readwave.c    writewave.c
oyabun% gcc -o readwave readwave.c
oyabun% ./readwave piano.wav > piano.txt
```

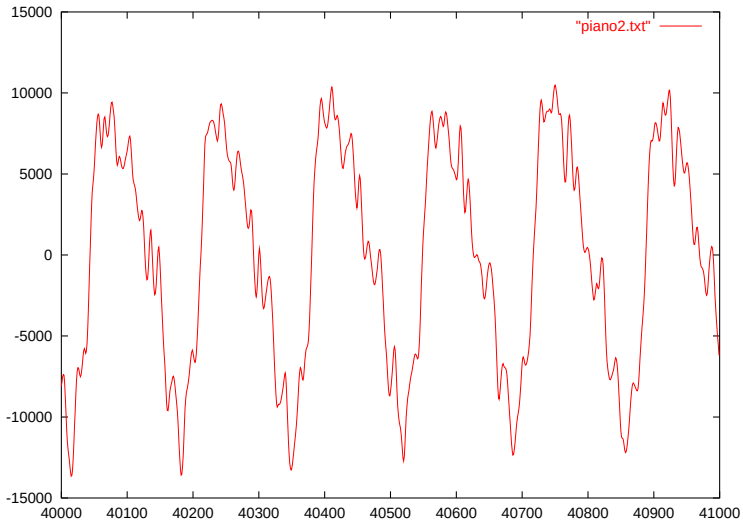
こうして出来た piano.txt は、cat や less, emacs などで読むことができる。

40000 行から 41000 行までを切り出して、gnuplot で表示

```
oyabun% gawk '(NR>=40000 && NR< 41000) {print NR,$1,$2}' piano.txt > piano2.txt
oyabun% gnuplot
gnuplot> plot "piano2.txt" with lines
```

[1] http://www.math.meiji.ac.jp/~mk/keisanki2-2005/matsuyama-wave/

[2] http://www.math.meiji.ac.jp/~mk/keisanki2-2005/readwave.c



piano.txt

```
# RIFF データのサイズ=1234584
# fmt データのサイズ=18
#  01 00 02 00 44 ac 00 00 10 b1 02 00 04 00 10 00
#  00 00
# 非圧縮 PCM です。
# ステレオです。
# サンプリング・レート (標本化周波数)=44100
# 1 秒当りのバイト数=176400
# ブロック境界=4
# ビット数/サンプル=16
# 拡張情報サイズ=0
# PAD チャンクがあります。# PAD チャンクのサイズ=4042
#  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 (中略)
#  00 00 00 00 00 00 00 00 00 00
# fact チャンクがあります。# fact データのサイズ=4
#  49 af 04 00
# data データのサイズ=1228068
#original file: piano.wav
#number of channels: 2
#sampling rate: 44100
#number of bits (per sample): 16
#number of samples: 307017
64 64
62 62
  (中略 --- 結構長い)
-114 -63
-111 -59
-112 -61
-109 -53
-108 -51
```

以上、素朴に音声データ (音圧の時間変化) をグラフに表示できた。
Java で Hello World! サウンド編[3] にある HelloWorldSound.java[4], HelloWorldRecorder.java[5]

[3] http://www.hellohiro.com/sound.htm
[4] http://www.hellohiro.com/src/HelloWorldSound.java
[5] http://www.hellohiro.com/src/HelloWorldRecorder.java

を参考にして、Java で実現したい。

```
while (nBytesRead != -1) {
        // オーディオストリームからデータを読み込みます
        nBytesRead = audioInputStream.read(abData, 0, abData.length);
        if (nBytesRead >= 0) {
          // オーディオデータをミキサーに書き込みます
          int nBytesWritten = line.write(abData, 0, nBytesRead);
        }
      }
```

abData は Byte の配列である。これを適当に数値に変換すれば良いであろう。

2 MitsuiWorld の改良

1. 等高線描画機能を加えた Mitsui_and_Ito.java というのがある (2004 年度卒研生伊藤秀範君作成)。ソース・プログラムや、サンプル・プログラム、実験用 WWW ページなど、http://www.math.meiji.ac.jp/~mk/labo/report/open/2004-itou-prog/ からリンクが張ってある。

2. 視点の方向（？）を変えられるような仕掛けを作る。 — これは現時点でイイ線行っている。

3. hideBirdViewInDisk(double [][]u, int nr, int nt, int h) というメソッドを作る。

最後の hideBirdViewInDisk() であるが、円盤領域で熱方程式、波動方程式を解いた時の可視化に使いたい (太鼓の振動とか)。これまでは、$r\theta u$ 空間でグラフを描いたり、**gnuplot** を使って描画していた。http://www.math.meiji.ac.jp/~mk/labo/2007/heat2d-disk.tar.gz

3 Java で FFT (1)

どうするのが一番良いかイマイチ分からないが、とりあえず Ooura's Mathematical Software Packages[6] にある「汎用 FFT (高速 フーリエ/コサイン/サイン 変換) パッケージ」[7] を使ってみる。

これはデータ数が 2 の冪でないとダメという制限はあるが、性能の高さには定評がある。

[6] http://www.kurims.kyoto-u.ac.jp/~ooura/index-j.html
[7] http://www.kurims.kyoto-u.ac.jp/~ooura/fft-j.html

```
oyabun% tar xzf fft.tgz
oyabun% cd fft
oyabun% ls
oyabun% cd sample1
oyabun% make
mathpc% ./test4g
data length n=? (must be 2^m)
1024
cdft err= 4.24682e-16
rdft err= 4.24682e-16
ddct err= 4.95724e-16
ddst err= 4.98625e-16
dfct err= 3.86626e-16
dfst err= 5.69098e-16
oyabun% cd ../sample2
oyabun% make
oyabun% ./pi_fft4g
oyabun% less pi.dat
```

例えば test4g.c を Java プログラムに書き換えると良い。

I 君はその作業をスタートしたようだが、上手く行ったかな？

FFTPACK を Java にするのは無理か？あるじゃん！jfftpack.tgz[8] — でも上手くコンパイル出来ない。なぜだろう。

- 「FFT についてのメモ」[9]
- FFTPACK[10]

4 波動方程式の差分近似 — I 君のプログラム

u_{tt}, u_{xx} を共に 2 階中心差分で近似してできた差分方程式を「波動方程式に対する差分法」[11] で説明してあるが、他にも色々な方法がある。

Friedrichs の差分法と呼ばれるものが有名である (藤沼祐一「波動方程式に対する差分法」[12] を見よ) が、今日の I 君が説明したようにして、時間について 1 階の方程式にして解くのも自然である。

ただ、今日のは、これまで卒研で学んだことの多くを「おいといて」という感じの内容でした。ツッコミを箇条書きにしておく。

[8] http://www.netlib.org/fftpack/jfftpack.tgz
[9] http://www.math.meiji.ac.jp/~mk/labo/text/fft-lecture.pdf
[10] http://www.math.meiji.ac.jp/~mk/labo/text/fftpack-index.pdf
[11] http://www.math.meiji.ac.jp/~mk/labo/text/wave.pdf
[12] http://www.math.meiji.ac.jp/~mk/labo/report/pdf/1998-fujinuma.pdf

(a) 0.00625 のようなマジック・ナンバーをプログラムに書いてはいけない。これは X (= 160) の逆数であろうから、dt=1.0/X; のようにすべきである。こうしておけば X の値を書き換えても正常に動作する。

(b) X や t という変数名は誤解されかねないので、不適当であろう。特に x や t は別の意味で使うことになる可能性が高い (偏微分方程式の記述に現れるのだから)。

(c) double [][]u = new double[X+2][T+2]; のように +2 とするのは、慎重なようで決してそうではない。正しく理解していれば double [][]u = new double[X+1][T+1]; で必要十分と言うことが分かるはず。

(d) そもそも double [][]u = new double[X+2][T+2]; のように 2 次元配列にするのは拙い。時間方向には 3 ステップ分だけあれば大丈夫。

ちょっと遊んでみる。

```
 1  /* fib1.c --- 配列を使う */
 2
 3  #include <stdio.h>
 4
 5  #define N (20)
 6
 7  int main(void)
 8  {
 9    int i;
10    int a[N+1];
11    a[0] = 1;
12    a[1] = 1;
13    for (i = 2; i <= N; i++)
14      a[i] = a[i - 1] + a[i - 2];
15    for (i = 0; i <= N; i++)
16      printf("a[%2d]=%5d\n", i, a[i]);
17    return 0;
18  }
```

```c
/* fib2.c --- 隣接３項の漸化式だから３つメモリーがあれば十分 */

#include <stdio.h>

#define N (20)

int main(void)
{
  int i;
  int am2, am1, a;
  am2 = 1;
  am1 = 1;
  printf("a[%2d]=%5d\n", 0, am2);
  printf("a[%2d]=%5d\n", 1, am1);
  for (i = 2; i <= N; i++) {
    a = am1 + am2;
    printf("a[%2d]=%5d\n", i, a);
    am2 = am1;
    am1 = a;
  }

  return 0;
}
```

```c
/* fib3.c --- ベクトル列になったときはコピーが面倒なので、添字でローテート
              まあ、リングバッファーみたいなもんです。 */

#include <stdio.h>

#define N (20)

int main(void)
{
  int i;
  int a[3];
  a[0] = 1;
  a[1] = 1;
  printf("a[%2d]=%5d\n", 0, a[0]);
  printf("a[%2d]=%5d\n", 1, a[1]);
  for (i = 2; i <= N; i++) {
    a[i%3] = a[(i-1)%3] + a[(i-2)%3];
    printf("a[%2d]=%5d\n", i, a[i%3]);
  }

  return 0;
}
```

```c
/* fib4.c --- 割り算 % をやめて */

#include <stdio.h>

#define N (20)

int main(void)
{
  int i,I,Im1,Im2;
  int a[3];
  a[0] = 1;
  a[1] = 1;
  printf("a[%2d]=%5d\n", 0, a[0]);
  printf("a[%2d]=%5d\n", 1, a[1]);
  Im1 = 1; Im2 = 0;
  for (i = 2; i <= N; i++) {
    I = Im1 + 1; if (I == 3) I = 0;
    a[I] = a[Im1] + a[Im2];
    printf("a[%2d]=%5d\n", i, a[I]);
    Im2 = Im1;
    Im1 = I;
  }

  return 0;
}
```

5 Zバッファ (1)

うすらぼんやりと次のように理解しています。

3 次元空間内の図形 Ω を、2 次元平面にピクセルを (長方形の形に) 並べたの D に「描く」のが目的である。Ω の各点を D のどこに写すかを指定する写像 $\varphi\colon \Omega \to D$ を用意する (透視変換と呼ぶ)。この φ は一対一でないため、本来「見える」はずのところを「見えない」はずのところで上書き描画して、見えなくなってしまうことを防ぐ必要がある。$p \in D$ に対して、$\varphi^{-1}\{p\}$ が 2 個以上の点を含むとき、p にどの $x \in \varphi^{-1}(\{p\})$ の像を描くのか、正しく決める必要がある。そのために D 内の各点にメモリーを持たせて、$p \in D$ に対して、$\varphi(x) = p$ となる $x \in \Omega$ が見つかったとき、...