画像工学 レポート

― 画像の高速フーリエ変換―

IE5 (9) 片岡 駿之介

2016 年 12 月 22 日

1　課題

画像処理は 2 次元の連続的な信号として考えることができる．そこで，この 2 次元の信号に対してフーリエ変換を行い，それによって得られた周波数スペクトルに対して周波数フィルタをかけることによって，画像処理を実現する方法がある．
今回の課題では，このフーリエ変換を高速化した FFT（Fast Fourier Transform）を用いて画像のフーリエ変換を行い，これによって得られたパワースペクトルを確認する．

2　処理の流れ

今回の演習では，PGM ファイルを以下の手順で処理していくこととした．

1. 画像を配列へ取り込む．
2. 画像を取り込んだ配列を実部を示す配列にコピーする．同時に，虚部を表す配列を 0 埋めする．
3. 実部・虚部を表す配列を引数として渡し，1 行ずつ fft 関数を実行する．
4. 実部・虚部を表す配列を転置する．
5. 再び，実部・虚部を表す配列を引数として渡し，1 行ずつ fft 関数を実行する．
6. 実部・虚部を表す配列を転置する．
7. 各ピクセルごとに，実部と虚部をそれぞれ 2 乗して足し合わせ，ルートを取ったものを出力する．
8. 出力によって得られたデータを dat ファイルに保存し，gnuplot で確認する．

3 ソースリスト

ソースコード 1　filter.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <ctype.h>
4   #include <math.h>
5
6   #define PI 3.14159265358979323846
7
8   FILE *fp;
9
10  int width;
11  int height;
12  int max_value;
13
14  int status;
15  char buffer[128];
16
17  static void make_sintbl(int n, double sintbl[])
18  {
19      int i, n2, n4, n8;
20      double c, s, dc, ds, t;
21
22      n2 = n / 2; n4 = n / 4; n8 = n / 8;
23      t = sin(M_PI / n);
24      dc = 2 * t * t; ds = sqrt(dc * ( 2 - dc));
25      t = 2 * dc; c = sintbl[n4] = 1; s = sintbl[0] = 0;
26      for (i = 1; i < n8; i++) {
27          c -= dc; dc += t * c;
28          s += ds; ds -= t * s;
29          sintbl[i] = s; sintbl[n4 - i] = c;
30      }
31      if (n8 != 0) sintbl[n8] = sqrt(0.5);
32      for (i = 0; i < n4; i++)
33          sintbl[n2 - i] = sintbl[i];
34      for (i = 0; i < n2 + n4; i++)
35          sintbl[i + n2] = - sintbl[i];
36  }
37
38  static void make_bitrev(int n, int bitrev[])
39  {
40      int i, j, k, n2;
41
42      n2 = n / 2; i = j = 0;
43      for ( ; ; ) {
44          bitrev[i] = j;
45          if (++i >= n) break;
46          k = n2;
47          while (k <= j) { j -= k; k /= 2; }
48          j += k;
49      }
50  }
51
52  int fft(int n, double x[], double y[])
```

```
{
    static int last_n = 0; /* 前回呼出し時の n */
    static int *bitrev = NULL; /* ビット反転表 */
    static double *sintbl = NULL; /* 三角関数表 */
    int i, j, k, ik, h, d, k2, n4, inverse;
    double t, s, c, dx, dy;

                                            /* 準備 */
    if (n < 0) {
        n = -n; inverse = 1; /* 逆変換 */
    } else inverse = 0;
    n4 = n / 4;
    if (n != last_n || n == 0) {
        last_n = n;
        if (sintbl != NULL) free(sintbl);
        if (bitrev != NULL) free(bitrev);
        if (n == 0) return 0; /* 記憶領域を解放 */
        sintbl = malloc((n + n4) * sizeof(double));
        bitrev = malloc(n * sizeof(int));
        if (sintbl == NULL || bitrev == NULL) {
            fprintf(stderr, "memory␣error\n"); return 1;
        }
        make_sintbl(n, sintbl);
        make_bitrev(n, bitrev);
    }
    for (i = 0; i < n; i++) { /* ビット反転 */
        j = bitrev[i];
        if (i < j) {
            t = x[i]; x[i] = x[j]; x[j] = t;
            t = y[i]; y[i] = y[j]; y[j] = t;
        }
    }
    for (k = 1; k < n; k = k2) { /* 変換 */
        h = 0; k2 = k + k; d = n / k2;
        for (j = 0; j < k; j++) {
            c = sintbl[h + n4];
            if (inverse) s = - sintbl[h];
            else s = sintbl[h];
            for (i = j; i < n; i += k2) {
                ik = i + k;
                dx = s * y[ik] + c * x[ik];
                dy = c * y[ik] - s * x[ik];
                x[ik] = x[i] - dx; x[i] += dx;
                y[ik] = y[i] - dy; y[i] += dy;
            }
            h += d;
        }
    }
    if (! inverse) /* 逆変換でないなら n で割る */
        for (i = 0; i < n; i++) { x[i] /= n; y[i] /= n; }
    return 0; /* 正常終了 */
}


void image_open(char *file_name) {
  if ((fp = fopen(file_name, "r")) == NULL) {
             printf("file␣open␣error!!\n");
```

```
110                 exit(-1); /* (3) エラーの場合は通常、異常終了する */
111         }
112       // printf("%s\n", file_name);
113     }
114
115     int main(int argc,char *argv[]) {
116
117       /* 入力コマンドのチェック */
118       if(argc == 1) {
119         printf("Usage␣:␣./fft␣<pgm␣file␣name>\n");
120         exit(0);
121       } else if (argc == 2) {
122         image_open(argv[1]);
123       }
124
125       /* ヘッダ取得部 */
126       int ch;
127
128       while (status < 3) {
129         ch = getc(fp);
130
131         if(ch == '#') { // コメントのスキップ
132           while (( ch = getc(fp)) != '\n')
133             break;
134         }
135
136         if(ch == 'P') { // マジックナンバーのチェック
137           if(getc(fp) != '5') {
138             printf("Magic␣Number␣is␣wrong.\n");
139             break;
140           } else {
141             // printf("Magic Number is P5\n");
142           }
143         }
144
145         if(isdigit((unsigned char)ch)) { // 数値取得部
146           buffer[0] = ch;
147           int i=1;
148           while(1) {
149               char c = getc(fp);
150               if(isdigit((unsigned char)c)) {
151                 buffer[i]=c;
152                 i++;
153               } else
154                 break;
155           }
156           buffer[i] = '\0';
157
158           switch (status) {
159             case 0: // width 取得部
160               width = atoi(buffer);
161               // printf("width=%d\n", width);
162               break;
163             case 1: // height 取得部
164               height = atoi(buffer);
165               // printf("height=%d\n", height);
166               break;
```

```
167           case 2: // max_value 取得部
168             max_value = atoi(buffer);
169             // printf("max_value=%d\n", max_value);
170             break;
171         }
172         status++; // 次のステータスへ
173       }
174     }
175
176     /* 画素値取得部 */
177     int image[width][height];
178
179     for(int i = 0; i < height; i++) {
180       for(int j = 0; j < width; j++) {
181         image[j][i] = (int)getc(fp);
182       }
183     }
184
185     double x[width][height];
186     double y[width][height];
187
188     for(int i = 0; i < height; i++) {
189       for(int j = 0; j < width; j++) {
190         x[j][i] = (double)image[j][i];
191         y[j][i] = 0.0;
192       }
193     }
194
195     for(int i = 0; i < height; i++) { //1 かいめの fft
196         fft(256, x[i], y[i]);
197     }
198
199     double x_tmp[width][height];
200     double y_tmp[width][height];
201
202     for(int i = 0; i < height; i++) { //1 かいめの転置
203       for(int j = 0; j < width; j++) {
204         x_tmp[i][j] = x[j][i];
205         y_tmp[i][j] = y[j][i];
206       }
207     }
208
209     for(int i = 0; i < height; i++) {
210       for(int j = 0; j < width; j++) {
211         x[i][j] = x_tmp[i][j];
212         y[i][j] = y_tmp[i][j];
213       }
214     }
215
216     for(int i = 0; i < height; i++) { //2 かいめの fft
217         fft(256, x[i], y[i]);
218     }
219
220     for(int i = 0; i < height; i++) { //2 かいめの転置
221       for(int j = 0; j < width; j++) {
222         x_tmp[i][j] = x[j][i];
223         y_tmp[i][j] = y[j][i];
```

```
224       }
225     }
226
227     for(int i = 0; i < height; i++) {
228       for(int j = 0; j < width; j++) {
229         x[i][j] = x_tmp[i][j];
230         y[i][j] = y_tmp[i][j];
231       }
232     }
233
234     for(int i = 0; i < height; i++) { //出力
235       for(int j = 0; j < width; j++) {
236         printf("%d␣%d␣%f\n", i, j, sqrt(pow(x[j][i], 2) + pow(y[j][i], 2)));
237       }
238       printf("\n");
239     }
240
241   }
```

4　結果

結果の確認には，配布された「cup.pgm」を用いた．

図 1　cup.pgm

以下に，FFT を行った際のパワースペクトルグラフを示す．

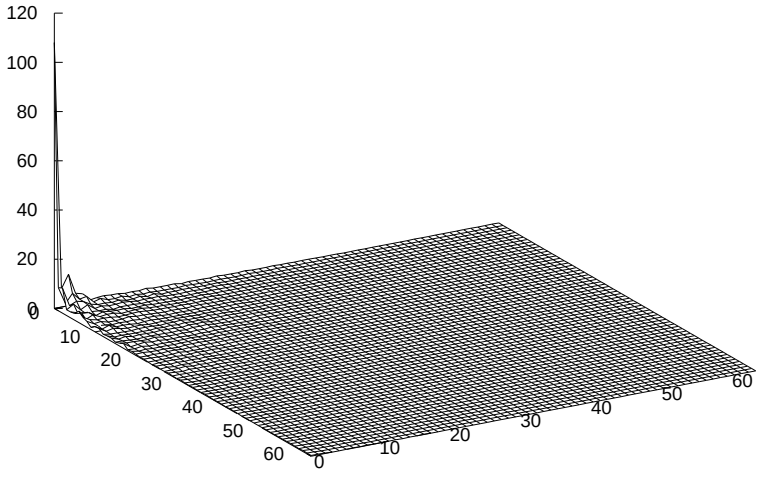

図 2　cup.pgm に対して FFT を行った際のパワースペクトル

5　考察

画像をフーリエ変換した際の周波数スペクトルは，低周波数成分に集中していることがわかった．
今まではピクセル操作で画像処理を行っていたが，周波数フィルタをかけるともっと簡単に実装できそうであると感じた．

6　感想

FFT が本当に高速で感動した．