Przetwarzanie i Kompresja Obrazów. Filtracja w dziedzinie widmowej. Implementacja

Aleksander Denisiuk (denisjuk@pja.edu.pl)
Polsko-Japońska Akademia Technik Komputerowych
Wydział Informatyki w Gdańsku
ul. Brzegi 55, 80-045 Gdańsk

2 maja 2016

Filtracja w dziedzinie widmowej. Implementacja

Najnowsza wersja tego dokumentu dostępna jest pod adresem
http://users.pja.edu.pl/~denisjuk/

FFTW

Podstawy

- The Fastest Fourier Transform in the West
- Zestaw legalnych (GPL 2+) procedur w C do obliczania Dyskretnej transformacji Fouriera
  - ciągów zespolonych, rzeczywistych
  - dowolnej długości
  - jedno-, dwu- i więcej wymiarowych
  - pozwala na obiczenia równoległe i rozproszone
- Zazwyczaj jest szybcza niż inne implementacje
  - Sun's Performance Library
  - IBM's ESSL
- Jest przenośna
- Powstała w MIT w 1997 roku
- Najnowsza wersja: 2.1.5, 12 lutego 2016
- Strona projektu: http://www.fftw.org/

Instalacja

- Zainstalować paczkę `fftw-dev`
  - dokumentacja: `fftw-doc`
- Można skompilować ze źródeł
- Liczne paczki do innych języków programowania:
  - C++
  - Lisp
  - Ruby
  - Python
  - MPI
  - Yorick
  - CUDA
  - ...

Wykorzystanie

- W kodzie (C, C++):

```
#include <fftw.h>
```

- W opcjach konsolidatora (linkera):

```
-lfftw -lm
```

- Dwa kroki:
  - utworzenie planu
  - obliczenie transformacji z wykorzystaniem planu
- Plan można zapisać i wykorzystać
  - nawet dla innych wymiarów ciągów

Typy danych


```
typedef double fftw_real;

typedef struct {
     fftw_real re, im;
} fftw_complex;

#define c_re(c)  ((c).re)
#define c_im(c)  ((c).im)
```

Co FFTW oblicza

- `FFTW_FORWARD`

$$X_k \mapsto Y_k = \sum_{l=0}^{n-1} X_l e^{-2\pi i k l/n}$$

- `FFTW_BACKWARD`

$$Y_k \mapsto X_k = \sum_{l=0}^{n-1} Y_l e^{2\pi i k l/n}$$

- w szczególności,

$$\texttt{FFTW\_BACKWARD}\big(\texttt{FFTW\_FORWARD}(X)\big) = nX$$

Jednowymiarowa transformacja


```
fftw_complex in[N], out[N];
  fftw_plan p;
  ...
  p = fftw_create_plan(N,
          FFTW_FORWARD, FFTW_ESTIMATE);
  ...
  fftw_one(p, in, out);
  ...
  fftw_destroy_plan(p);
```

- Odwrotna transformacja: `FFTW_BACKWARD`
- Inna strategia tworzenia planu: `FFTW_MEASURE`

Dwuwymiarowa transformacja


```
fftw_complex in[L][M], out[L][M];
  fftwnd_plan p;
  ...
  p = fftw2d_create_plan(L, M,
          FFTW_FORWARD, FFTW_ESTIMATE);
  ...
  fftwnd_one(p, &in[0][0], &out[0][0]);
  ...
  fftw_destroy_plan(p);
```

- Inny typ planu: `fftwnd_plan`
- Inna procedura tworzenia planu: `fftw2d_create_plan`
- Inaczej przekazujemy tablicę danych: `&in[0][0]`
- Inna funkcja do obliczeń: `fftwnd_one`

Obliczenie in-place


```
fftw_complex in[L][M], out[L][M];
  fftwnd_plan p;
  ...
  p = fftw2d_create_plan(L, M,
          FFTW_FORWARD,
          FFTW_ESTIMATE | FFTW_IN_PLACE);
  ...
  fftwnd_one(p, &in[0][0], NULL);
  ...
  fftw_destroy_plan(p);
```

- Tablica wejśiowa jest nadpisywana
- Mniej pamięci
- Na ogół jest szybciej

Tablice dynamiczne


```
fftw_complex *an_array;
an_array = (fftw_complex *)
  malloc(L * M * sizeof(fftw_complex));
```

- Tablice są przechowywane wierszami (row major)
- $a_{00} \quad a_{01} \quad \ldots a_{0\,L-1} \quad a_{10} \quad \ldots \quad a_{1\,L-1} \quad a_{M-1\,L-1}$
- $a_{l\,m} =$ `an_array[m+l*L]`

```
free(an_array);
```

```
fftw_complex in[L][M][N];
  fftwnd_plan p;
  ...
  p = fftw3d_create_plan(L, M, N,
          FFTW_FORWARD,
          FFTW_MEASURE | FFTW_IN_PLACE);
  ...
  fftwnd_one(p, &in[0][0][0], NULL);
  ...
  fftw_destroy_plan(p);
```

Wymiar D

```
fftw_complex* in;
fftwnd_plan p;
const int dimensions = [K, L, ..., M, N];
  ...
  in = (fftw_complex *)
    malloc( K * L *...* M * *N * sizeof(fftw_complex));
//  in[ n + N * (m + M * (... + L * k))] = a_{k l ... m n}
  ...
  p = fftwnd_create_plan(D, dimensions
          FFTW_FORWARD,
          FFTW_ESTIMATE | FFTW_IN_PLACE);
  ...
  fftwnd_one(p, in, NULL);
  ...
  fftw_destroy_plan(p);
```

Studium przypadku

Filtr Gaussowski w dziedzinie widmowej

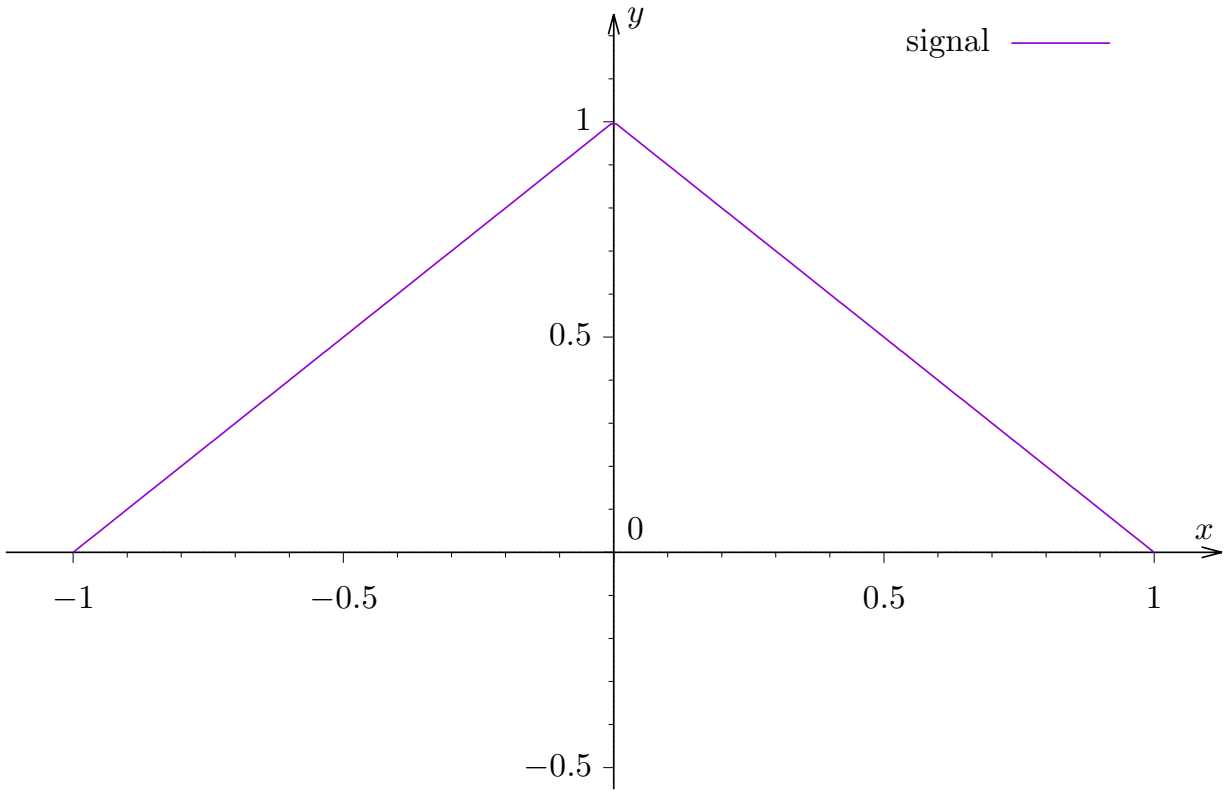
- W dziedzinie przestrzennej $f(x,y) \mapsto f(x,y) * \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$
- Transformacja Fouriera:
  $$\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \mapsto \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{\infty} e^{-\frac{x^2}{2\sigma^2}} e^{-2\pi i x u}\, dx = e^{-2\pi^2 u^2 \sigma^2}$$
- W dziedzinie widmowej $F(u,v) \mapsto F(u,v) \cdot e^{-2\pi^2(u^2+v^2)\sigma^2}$

Ciągła a dyskretna transformacja Fourira

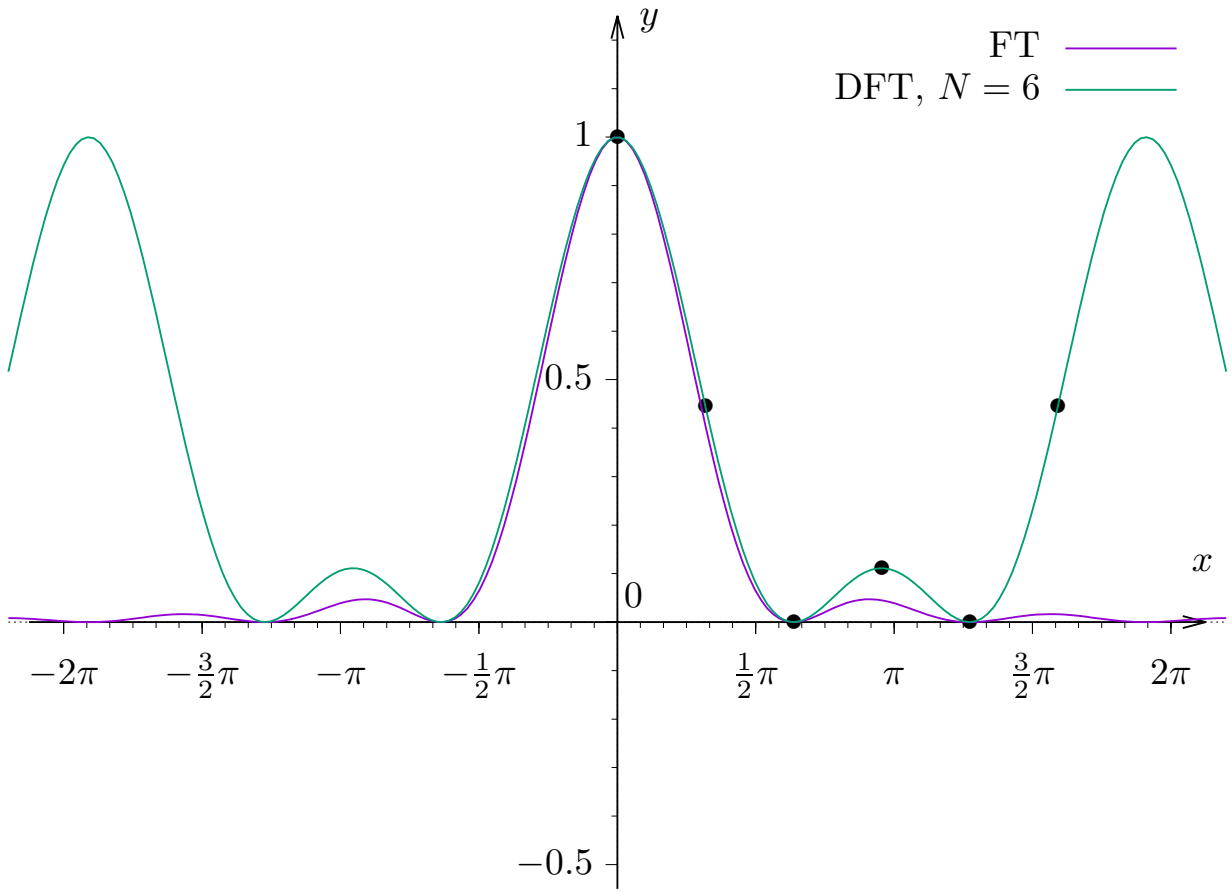
- DTF różni się od ciągłej transformacji Fouriera
- DTF jest okresową: $\sum_{k=0}^{N-1} f\left(\frac{k}{N}\right) e^{-2\pi i \frac{ku}{N}}$
- Ciągła — nie: $\int_{-\infty}^{\infty} f(x) e^{-2\pi i x u}\, dx$
- Potrzebne jest skalowanie
  - może zależeć od filtru

Przykładowy sygnal

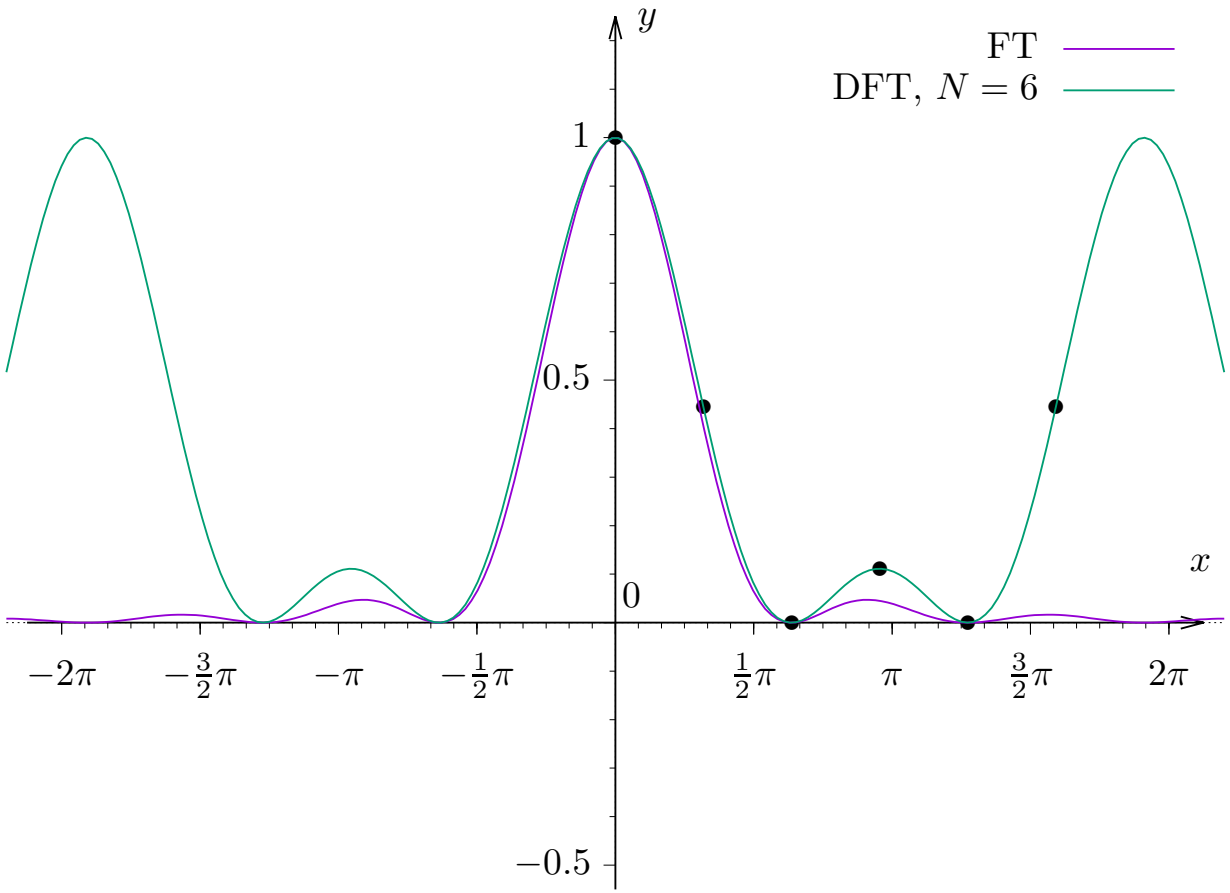
- $f(x) = 1 - |x|$ dla $x \in [-1, 1]$



- Ciągła: $\hat{f}(u) = \frac{2-2\cos \pi u}{(\pi u)^2}$
- Dyskretna (dla $N = 2n$): $\hat{f}_u = \frac{1}{n}\left(1 + 2\sum_{k=1}^{n-1} \frac{n-k}{n} \cos \frac{\pi k u}{n}\right)$



- $\hat{f}_k$ odpowiada częstotliwości $\frac{k}{N}$ dla $k \leqslant N/2$
- oraz częstotliwości $-\frac{N-k}{N}$ dla $k > N/2$



- Jak wykle:
  - wczytywanie pliku
  - opracowanie parametrów wierza poleceń
  - zapisywanie pliku

Przydział pamięci na tablice dla DTF trzech kanałów


```
ftr = (fftw_complex *)
    malloc(sizeof(fftw_complex)*width*height);
ftg = (fftw_complex *)
    malloc(sizeof(fftw_complex)*width*height);
ftb = (fftw_complex *)
    malloc(sizeof(fftw_complex)*width*height);
```

Wypełnienie tablic


```
for (y=0; y<height; y++) {
  JSAMPROW row = row_pointers[y];
  for (x=0; x<width; x++) {
      JSAMPROW ptr = &(row[x*3]);
      j= y + x*height;
      ftr[j].re = ptr[0];
      ftr[j].im = 0;
      ftg[j].re = ptr[1];
      ftg[j].im = 0;
      ftb[j].re = ptr[2];
      ftb[j].im = 0;
  }
}
```

```
if(sigma>0){
        gauss(sigma);
}
```

Kopiowanie wyników do trzech kanałów


```
for (y=0; y<height; y++) {
  JSAMPROW new_row = new_pointers[y];
  for (x=0; x<width; x++) {
      JSAMPROW new_ptr = &(new_row[x*3]);
      j= y + x*height;
      new_ptr[0] = get_byte(ftr[j].re);
      new_ptr[1] = get_byte(ftg[j].re);
      new_ptr[2] = get_byte(ftb[j].re);
  }
}
```

Zwolnienie pamięci


```
free(ftr);
free(ftg);
free(ftb);
```

Procedura pomocnicza


```
JSAMPLE get_byte(fftw_real x){
  JSAMPLE sample = x+0.5;
  if (sample < 0) return 0;
  if (sample > 255) return 255;
  return sample;
}
```

```
void gauss(fftw_real sigma){
  fftwnd_plan plan;
  fftw_real scale;
  fftw_real d2, s2, dd;
  fftw_real X, Y;
  int x, y, j;

  scale = (fftw_real)width * (fftw_real)height;
  s2 = M_PI*M_PI*2*sigma*sigma;
```

Prosta transformacja


```
plan = fftw2d_create_plan(
  width, height,
  FFTW_FORWARD,
  FFTW_ESTIMATE | FFTW_IN_PLACE);
fftwnd_one(plan, ftr, NULL);
fftwnd_one(plan, ftg, NULL);
fftwnd_one(plan, ftb, NULL);

fftwnd_destroy_plan(plan);
```

```
for (y=0; y<height; y++) {
  Y = (fftw_real)min(y, height-y)/(fftw_real)height;
  for (x=0; x<width; x++) {
      X = (fftw_real)min(x, width-x)/(fftw_real)width;
      d2 = X*X + Y*Y;
      j= y + x*height;
      dd = exp(d2*s2);
      ftr[j].re /= scale*dd;
      ftg[j].re /= scale*dd;
      ftb[j].re /= scale*dd;
      ftr[j].im /= scale*dd;
      ftg[j].im /= scale*dd;
      ftb[j].im /= scale*dd;
  }
}
```

Odwrotna transformacja


```
plan = fftw2d_create_plan(
  width, height,
  FFTW_BACKWARD,
  FFTW_ESTIMATE | FFTW_IN_PLACE);
fftwnd_one(plan, ftr, NULL);
fftwnd_one(plan, ftg, NULL);
fftwnd_one(plan, ftb, NULL);

fftwnd_destroy_plan(plan);
```

Procedura pomocnicza


```
int min(int x, int y){
    return x < y ? x : y;
}
```

Wynik