Filtrace snímků ve frekvenční oblasti

Rychlá fourierova transformace

semestrální práce z předmětu KIV/ZVI

zpracoval:

Jan Bařtipán

A03043

bartipan@students.zcu.cz

Obsah

Úvod

Fourierova transformace je pojmenována po Jean Baptiste Joseph Fourierovi (1768–1830), francouzském matematikovi a fyzikovi. Jde o matematický nastroj, s jehož pomocí lze vyjádřit jakýkoliv signál jako součet sinových vln s různou frekvencí.
Fourierova transformace F převádí tedy posloupnost N vzorků (označme ji $x=\{x_0, x_1, \ldots, x_{N-1}\}$) na jinou posloupnost N vzorků (označme ji $X=\{X_0, X_1, \ldots, X_{N-1}\}$). Tato transformace se nazývá *dopřednou Fourierovou transformací*. K této transformaci existuje transformace *inverzní* F^{-1}, která analogicky posloupnost X převede na posloupnost x.

$$F(x)=X$$
$$F^{-1}(X)=f(X)=x$$

Prvky $x_k \in x$ se nazývají prvky *v časové oblasti.* Prvky $X_k \in X$ se označují za prvky v *oblasti frekvenční.* Prvky posloupností x, X jsou komplexní čísla.

$$x_k \in \mathbb{C}$$
$$X_k \in \mathbb{C}$$

Diskrétní Fourierova transformace v 1D (DFT)

Mějme diskrétní signál N vzorků, vyjádřených posloupností hodnot $f_n, n \in \langle 0; N-1 \rangle$. Tento signál pomocí Fourierovy transformace může převést z časové oblasti do frekvenční takto:

$$F_k = \sum_{n=0}^{N-1} f_n \cdot e^{\frac{-i \cdot 2\pi \cdot k n}{N}}$$

Této transformaci se říká dopředná. Obdobně lze převést signál z frekvenční oblasti do časové takto:

$$f_k = \frac{1}{N} \cdot \sum_{n=0}^{N-1} F_n \cdot e^{\frac{i \cdot 2\pi \cdot k n}{N}}$$

Je patrné, že kdybychom implementovali výpočet fourierovy transformace podle tohoto vzorce, tento výpočet by měl složitost $O(n^2)$.

Rychlá Fourierova transformace (FFT)

V roce 1965 objevili[1] J. W. Cooley a J. W. Tukey metodu výpočtu Fourierovy transformace s výpočetní složitostí $O(n \cdot \log_2 n)$. Princip tohoto urychlení spočívá v rozkladu posloupnosti na podposloupnosti s lichými a sudými prvky původní posloupnosti. Koeficienty Fourierovy transformace se vypočtou z Fourierových transformací vzniklých podposloupností. Platí totiž [Wolfram]:

1 Ve skutečnosti tato metoda výpočtu byla objevena poprvé německým matematikem Karlem Fridrichem Gaussem (1777–1855) o více než století dříve.

$$F_k = \sum_{n=0}^{N-1} f_n \cdot e^{\frac{-i \cdot 2\pi \cdot kn}{N}}$$

$$F_k = \sum_{n=0}^{N/2-1} f_{2n} \cdot e^{\frac{-i \cdot 2\pi \cdot k2n}{N}} + \sum_{n=0}^{N/2-1} f_{2n+1} \cdot e^{\frac{-i \cdot 2\pi \cdot k(2n+1)}{N}}$$

$$F_k = \sum_{n=0}^{N/2-1} f_n^e \cdot e^{\frac{-i \cdot 2\pi \cdot kn}{N/2}} + e^{\frac{-j \cdot 2\pi}{N} \cdot k} \cdot \sum_{n=0}^{N/2-1} f_n^o \cdot e^{\frac{-i \cdot 2\pi \cdot kn}{N/2}}$$

$$F_k = F_k^e + e^{\frac{-j \cdot 2\pi}{N} \cdot k} \cdot F_k^o$$

Tento rozklad posloupností na podposloupnosti lze provádět rekurzivně, dokud nezískáme podposloupnost obsahující jen jeden prvek. Koeficient Fourierovy transformace této podposloupnosti je sám tento prvek. Abychom bez problémů mohli rekurzivně rozkládat posloupnosti, je třeba, aby signál, který chceme transformovat, měl právě 2^n vzorků.

In-place implementace algoritmu FFT

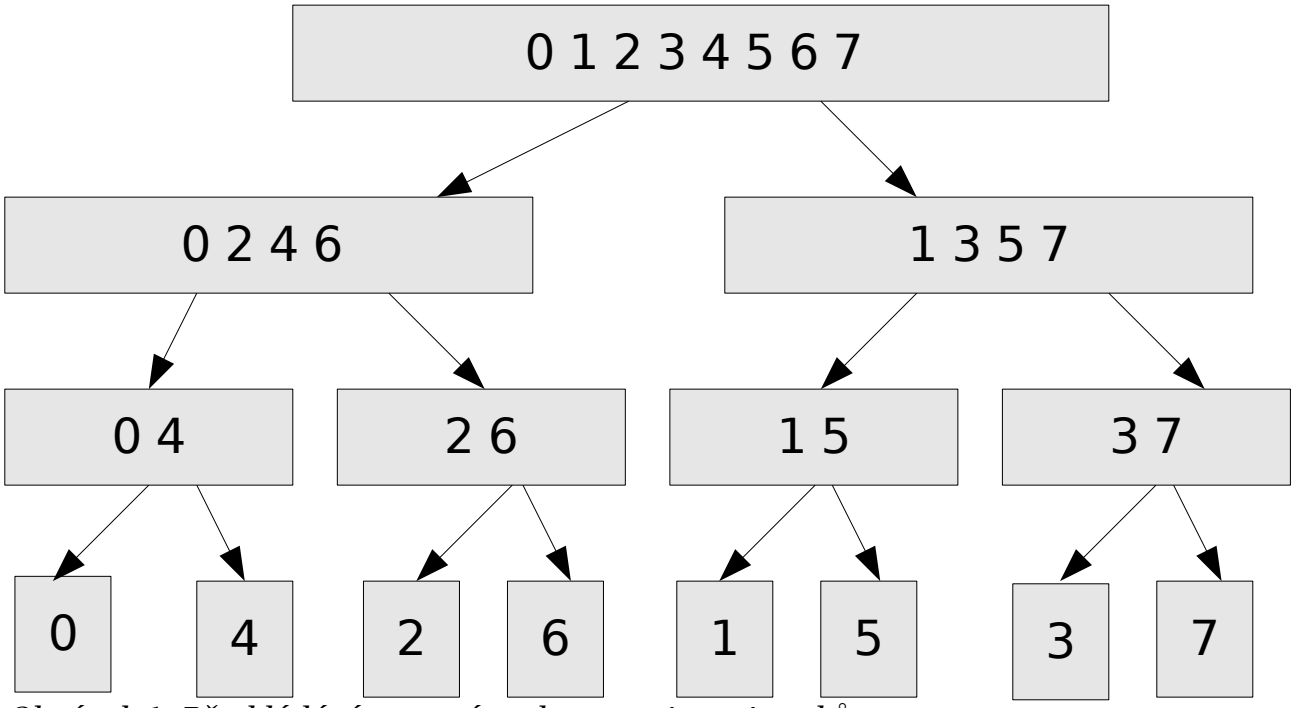
Algoritmus FFT se nejčastěji implementuje jako in-place algoritmus. To znamená, že pro výpočet FFT si vystačíme pouze s operační pamětí pro uchování prvků posloupnosti.

```
void fft(Array& arr, bool forward)
{
```

Přeřazení vstupní posloupnosti

Abychom mohli pohodlně provádět in-place implementaci FFT, před samotným výpočtem přeřadíme vstupní posloupnost. Tímto dostaneme v posloupnosti vždy vedle sebe dva bloky, které tvoří v určitém stupni lichou a sudou podposloupnost. Princip přeřazení posloupnosti o osmi prvcích je naznačenna obrázku 1.

```
  size_t n = arr.size();

  // Do the bit reversal
  u32 i2 = n >> 1;
  u32 j = 0;
  for (u32 i=0; i < n-1; i++)
  {
     if (i < j)
     {
        Complex tmp = arr[i];
        arr[i] = arr[j];
        arr[j] = tmp;
     }
     u32 k = i2;
     while (k <= j)
     {
        j -= k;
        k >>= 1;
     }
     j += k;
  }
```



Obrázek 1: Přeskládání vstupní posloupnosti osmi prvků.

Původní index		*Nový index*	
Dec.	***Bin.***	***Dec***	***Bin***
0	000	0	000
1	001	4	100
2	010	2	010
3	011	6	110
4	100	1	001
5	101	5	101
6	110	3	011
7	111	7	111

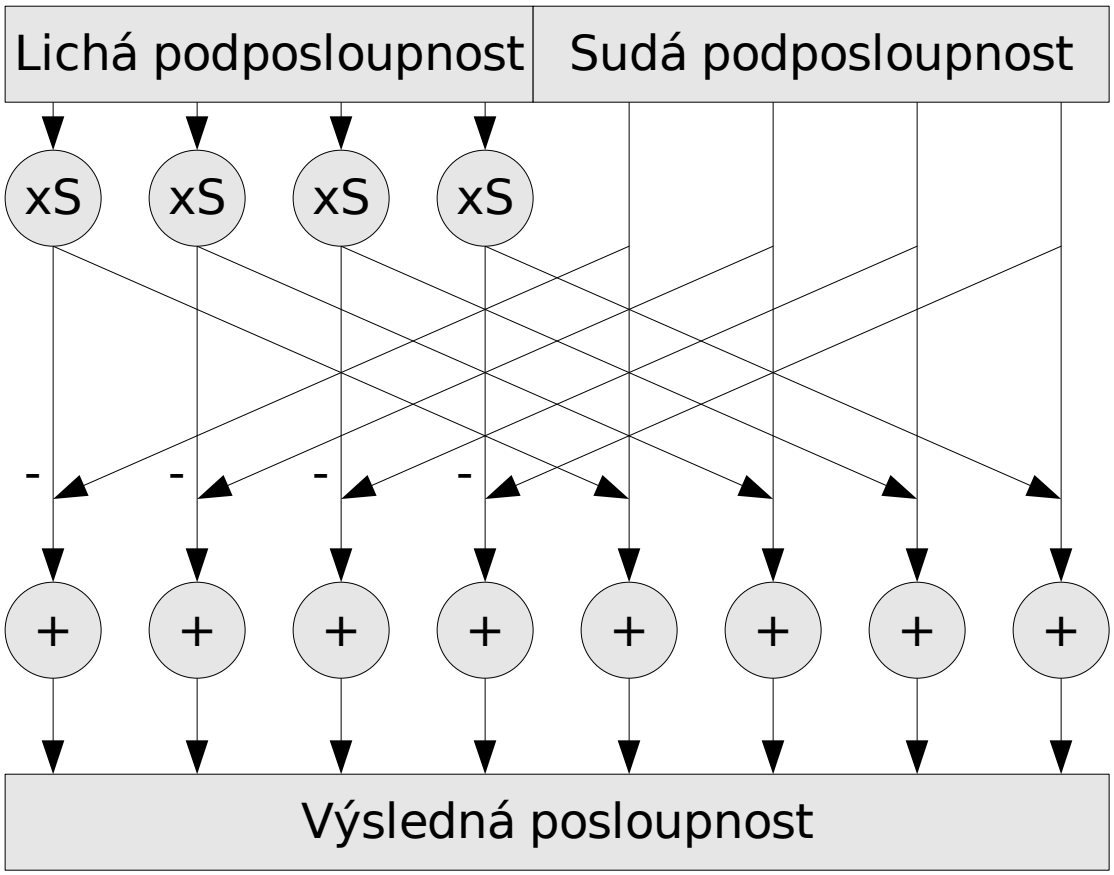
Tabulka 1: Původní a nový index jednotlivých prvku posloupnosti.

Princip výpočtu nového indexu prvku nám bude zřejmější, pokud si přepíšeme do tabulky hodnoty původního a nového indexu prvku. Při vyjádření hodnot indexů ve dvojkové soustavě si povšimněte, že hodnota nového indexu vznikne překlopením jednotlivých bitů kolem prostředního bitu (vzhledem k velikosti posloupnosti N).

Samotný výpočet pak provádí sestavování podposloupností a dopočítání koeficientů Fourierovy transformace.

Motýlkový algoritmus

Samotné skládání jednotlivých podposlouností provádí specializovaný algoritmus tzv. Motýlkový algoritmus. Ten provádí postupné skládání prvků z liché a sudé posloupnosti do dvou po sobě jdoucí prvků nové posloupnosti.



Obrázek 2: Motýlkový algoritmus na posloupnosti N=8

Jak je vidět na obrázku 2, každý prvek výsledné posloupnosti vznikne z příslušného prvku liché a sudé posloupnosti, tedy:

$$F_k = F_k^e + e^{\frac{-j \cdot 2\pi}{N} \cdot k} \cdot F_k^o$$

$$S_k = e^{\frac{-j \cdot 2\pi}{N} \cdot k}$$

```
// do computation
u32 l1;
u32 l2 = 1;
u32 m = static_cast<u32>(log(n)/log(2));

// sign of exponent
f64 sign = (forward) ? -1.0 : 1.0;
// base component of exponent of C
f64 cExpBase = sign * M_PI;

// initial
//Complex c = -1.0;
// loop for each level
for (u32 l = 0; l < m; l++)
{
   // loop for each sub DFT element
   // count of sub frequency spectras
   l1 = l2;
   // sub DFT stride
   l2 <<= 1;
```

```
      // W ^ 0
      Complex w(1,0);
      // W ^ 1
      Complex c(cos(cExpBase/l1), sin(cExpBase/l1));

      for (u32 j = 0; j < l1; j++)
      {
         // loop for each butterfly (sub DFT)
         for (u32 i = j; i < n; i += l2)
         {
            u32 i2 = i + l1;
            Complex t = w * arr[i2];
            arr[i2] = arr[i] - t;
            arr[i] += t;
         }

         // compute new w
         w *= c;
      }
   }
```

Normalizace zpětné Fourierovy transformace

Nakonec nesmíme u zpětné Fourierovy transformace zapomenout normalizovat celou posloupnost:

```
   // normalize
   if (!forward)
   {
      arr /= static_cast<f64>(n);
   }
}
```

Fourierova transformace ve 2D

Algoritmus výpočtu DFT ve 2D je dán jako výpočet koeficientů dle vzorce:

$$X_{k,l}=\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} x_{m,n}\cdot e^{-j2\pi(\frac{km}{M}+\frac{ln}{N})}$$

pro dopřednou Fourierovu transformaci a pro inverzní:

$$x_{k,l}=\frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} X_{m,n}\cdot e^{-j\pi(\frac{km}{M}+\frac{\ln}{N})}$$

Je zřejmé, že takto formulovaný výpočet DFT má složitost $O(n^4)$.

Pro výpočet dvou rozměrné Fourierovy transformace lze využít algoritmus výpočtu Fourierovy transformace v 1D. Snadno lze dokázat, že:

$$\begin{aligned}\xi_{1\ldots N,i}&=F_{1\mathrm{D}}(x_{i,1\ldots M})\\ X_{1\ldots N,i}&=F_{1\mathrm{D}}(\xi_{1\ldots N,i})\end{aligned},$$

kde x je matice vzorků v časové oblasti, $x_{i.1\ldots M}$ je sloupec (resp. řádek) v matici x , ξ je matice složená po řádcích z jednotlivých transformovaných sloupců ve frekvenční oblast a obdobně $X_{1\ldots M,i}$ je sloupec (resp. řádek) v této matici. Nic nám tedy nebrání pro výpočet Fourierovy transformace v 2D využít algoritmus FFT pro

výpočet 1D Fourierovy transformace.

Pro úplnost uvádím zdrojový kód pro 2D FFT:

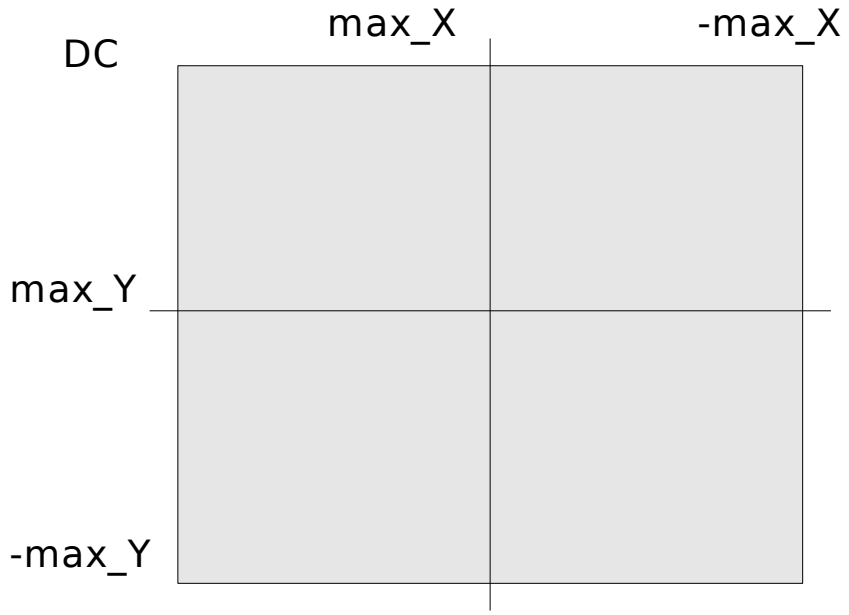
```
void fft2(Array& arr, u32 w, bool forward)
{
   u32 h = arr.size() / w;
   // perform 1D FFT for each row
   Array rArr(w);
   for (u32 i = 0; i < h; i++)
   {
      for (u32 j = 0; j < w; j++)
      {
         rArr[j] = arr[i * w + j];
      }
      fft(rArr, forward);
      for(u32 j = 0; j < w; j++)
      {
         arr[i * w + j] = rArr[j];
      }
   }

   // perform 1D FFT for each column
   Array cArr(h);
   for (u32 i = 0; i < w; i++)
   {
      for (u32 j = 0; j < h; j++)
      {
         cArr[j] = arr[j * w + i];
      }
      fft(cArr, forward);
      for(u32 j = 0; j < h; j++)
      {
         arr[j * w + i] = cArr[j];
      }
   }
}
```

Posunutí výstupu FFT

Výstupem dvourozměrné FFT je matice, která svými rozměry odpovídá vstupní matici. Každý prvek této matice vyjadřuje velikost afázi frekvence, která odpovídá pozici prvku v matici. Rozložení frekvencí v matici je znázorněno na obrázku 3.

Na tomto obrázku DC představuje (prvek na pozici 0,0) tzv. stejnosměrnou složku (součet všech prvků původního obrázku), X_max označuje prvky matice, které obsahují koeficienty pro největší frekvenci ve vodorovném směru, obdobně Y_max označuje prvky matice s největší frekvencí ve svislém směru.

Aby se nám lépe pracovalo s výstupní maticí, přemístíme jednotlivé kvadranty tak, aby DC složka byla uprostřed matice, koeficient frekvence (–X_max, –Y_max) byl v levém horním rohu a koeficient frekvence (X_max, Y_max) v pravém dolním rohu.



Obrázek 3: Rozložení frekvencí na jednotlivých prvcích výstupní matice

Reprezentace výsledků Fourierovy transformace

Výstupem dopředné Fourierovy transformace je posloupnost X, kde jednotlivé prvky této posloupnosti $X_i \in \mathbb{C}$. Jednotlivé prvky X_i vyjadřují zastoupení i–té frekvence v posloupnosti x. Pro snazší pochopení významu jednotlivých prvků, se používá polárního vyjádření koeficientu. Z něj lze zjistit velikost amplitudy frekvence (M_i), nebo její fázový posun (ϕ_i):

$$\phi_i = atan(\frac{\Re X_i}{\Im X_i})$$

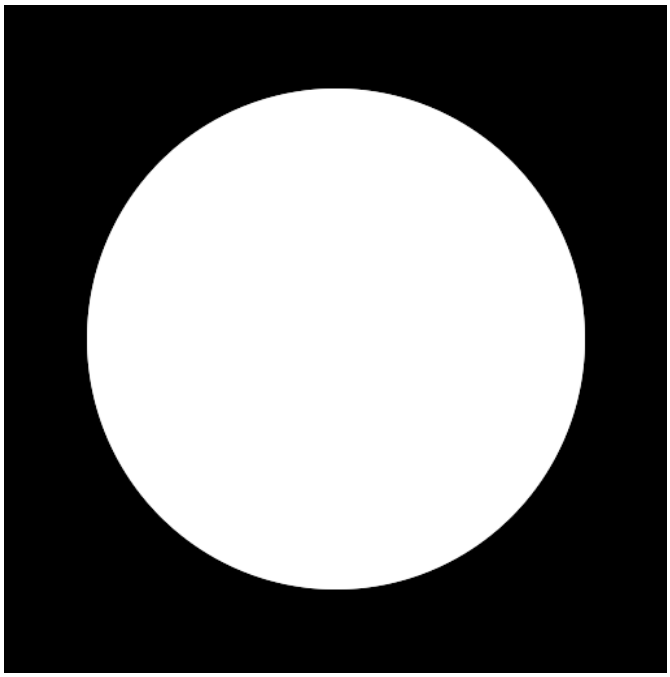
$$M_i = |X_i| = \sqrt{\Re X_i^2 + \Im X_i^2}$$

Filtrování pomocí Fourierovy transformace

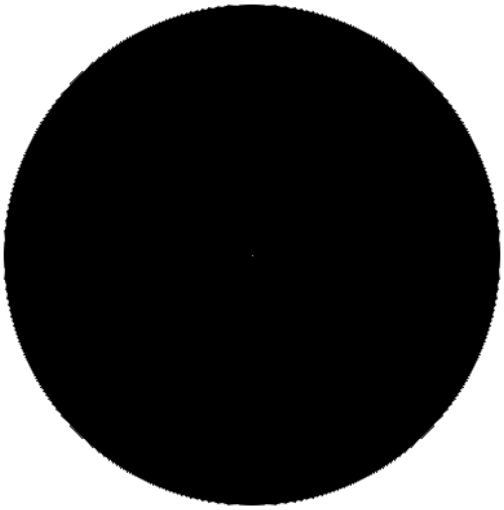
Pomocí Fourierovy transformace lze provádět filtrování ve frekvenčníoblasti. Princip fungování takovýchto filtrů je v těchto krocích:

1) transformace z časové do frekvenční oblasti
2) aplikování masky, na prvky ve frekvenční oblasti
3) zpětná transformace z frekvenční do časové oblasti

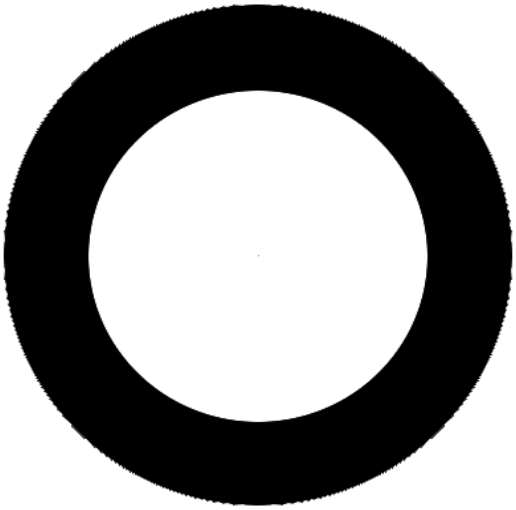
Aplikace masky v kroku dvě změníme (nastavíme na nulu) některé členy ve frekvenční oblasti. Maska tedy určuje pozice bodů, které budou nastaveny na nulu a pozice bodů, které zůstanou nezměněny. Při aplikování masky je třeba se vyvarovat odstranění stejnosměrné složky. Ta má význam těžiště a při její ztrátě dojde k citelné deformaci signálu.



Obrázek 4: Maska dolní propusti



Obrázek 5: Maska horní propusti

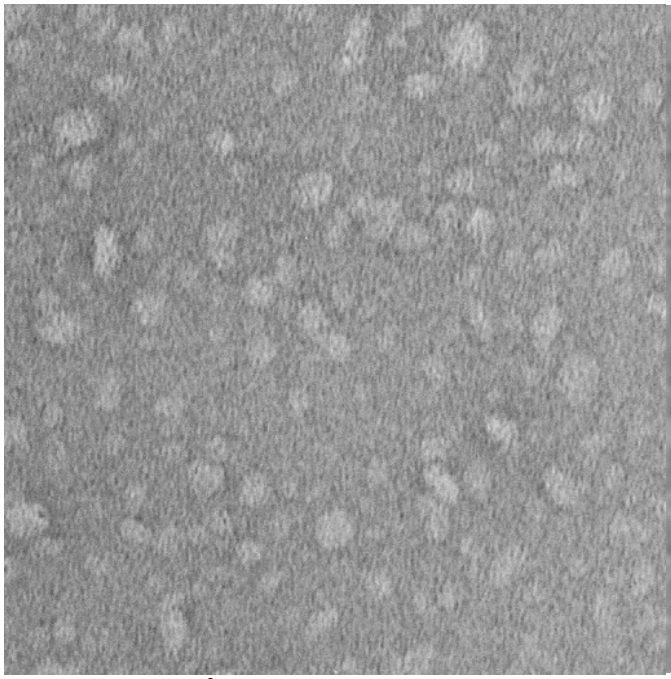


Obrázek 6: Maska pásmové propusti

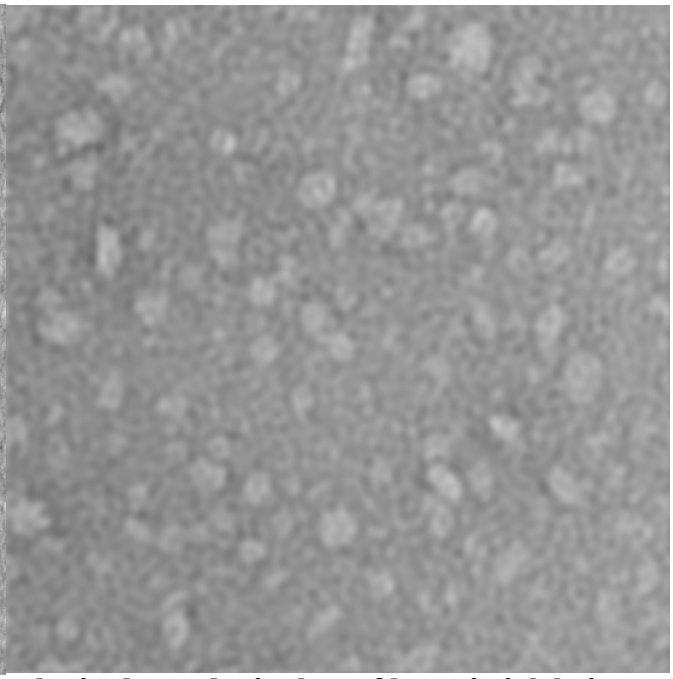
Implementace

Součástí této semestrální práce je i implementace rychlé Fourierovy transformace a její použití pro realizaci filtru dolní propustina obrázek. Program je napsánv jazyce C++ s použitím knihovny STL (třída *valarray*). Aplikace dále využívá knihovnu DevIL

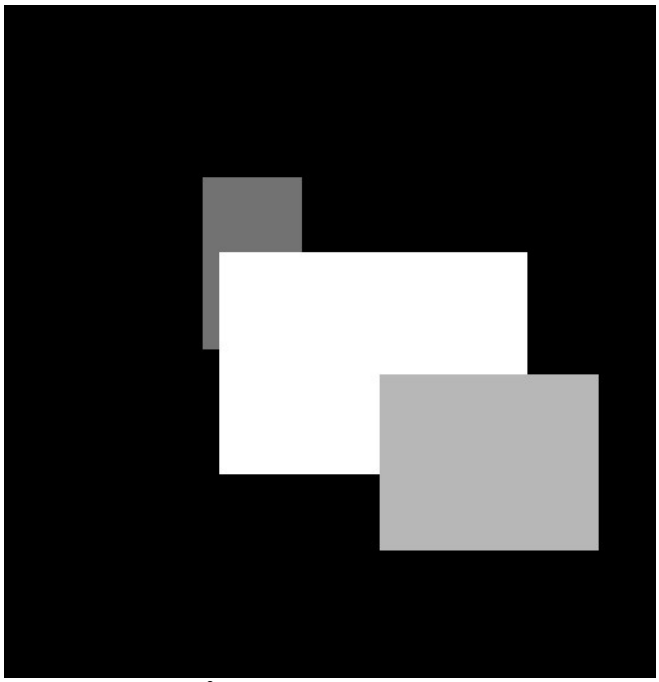
(http://openil.sf.net/) pro práci s obrázky. Program byl vyvíjen v prostředí GNU/Linux s použitím překladače GNU C++ (*g++*), nicméně jsme striktně používal přenositelné knihovny, a proto by překlad pro jiný operační systém neměl být problém. Tato implementace je přístupná na adrese http://home.zcu.cz/~bartipan/zvifft.tar.bz2



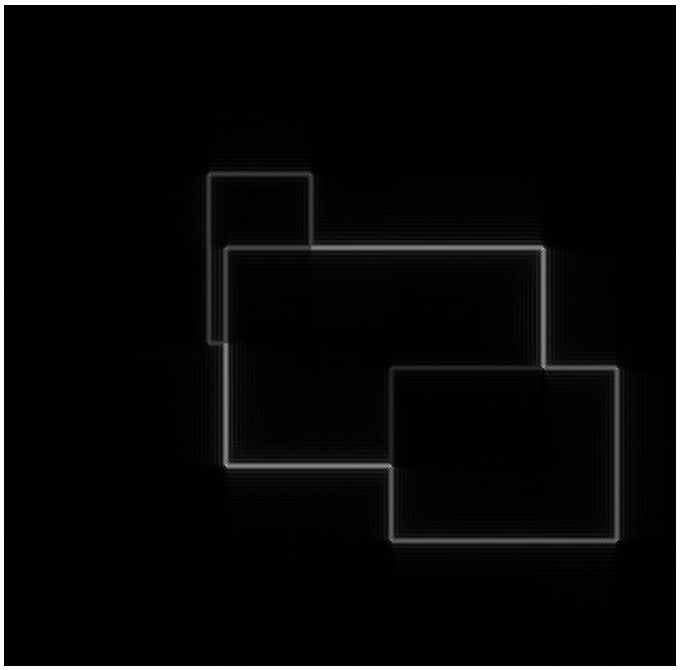
Obrázek 7: Původní obrázek



Obrázek 8: Obrázek po filtrování dolní propustí



Obrázek 9: Původní obrázek



Obrázek 10: Obrázek po filtraci horní propustí

Závěr

V této práci jsem se pokusil objasnit algoritmus rychlé Fourierovy transformace. Uvedl jsem matematický vztah pro složení sudé a liché podposloupnosti, který je základem FFT algoritmu. Dále jsem popsal postup, jak vypočíst Fourierovu transformaci ve 2D s použitím algoritmu FFT pro 1D transformaci. Na závěr jsem naznačil jak tento algoritmus použít k realizaci dolní propusti pro odstranění šumu z obrázku a horní propusti pro detekci hran. Implementace filtru dolní propusti v jazyce C++ je přístupná na adrese http://home.zcu.cz/~bartipan/zvi/fft.tar.bz2.

Reference

1. [DSPGuide] – The Scientist guide to DSP, http://www.dspguide.com/pdfbook.htm
2. [Wolfram] – http://mathworld.wolfram.com/FastFourierTransform.html
3. [RecipC] – Numerical Recipes in C, http://www.library.cornell.edu/nr/bookcpdf.html
4. [FFT2] – http://astronomy.swin.edu.au/~pbourke/other/fft2d/
5. [Impl] – How to implement the FFT algorithm – http://www.codeproject.com/cpp/howtofft.asp