

# FFT (Fast Fourier Transform)

## Plugins in ImageJ

Wolfgang Konen

TH Köln

2006 – 2021

**Abstract** *Dieser kurze Technical Report erläutert die Benutzung der FFT-Plugins für ImageJ. Es werden die Klassen und Methoden soweit erläutert, dass man die durch die FFT-Plugins bereitgestellte Funktionalität in eigene Java-Plugins einbinden kann. Ein Beispiel zum "idealen" Tiefpassfilter wird gegeben.*

Überblick.....	1
Vor- und Nachteile Built-In FFT (mit FHT) .....	1
Vor- und Nachteile FFTJ Plugin.....	1
Built-In FFT und FHT im Detail .....	2
FFTJ Plugin im Detail.....	5
API-Kurz-Doku zu FFTJ .....	5
Beispiel "Idealer" Tiefpass .....	6
Fazit.....	7

### Überblick

Es gibt in ImageJ mindestens zwei Methoden, eine FFT zu machen:  
Bulit-In FFT (mit FHT) oder Plugin FFTJ.

#### Vor- und Nachteile Built-In FFT (mit FHT)

Menu "Process – FFT" oder ij.plugin.FFT

- Vorteil: nutzt Fast Hartley Transform (FHT) und ist sehr schnell
- Vorteil: kann für verschiedenste Bildtypen (Byte-, Short-, Float-, ColorProcessor sowie Stack mit zwei Bildern (Real+Imag-Teil)) durchgeführt werden.
- Geht schnell für verschiedenste Bildgrößen: vor Übergang in den Frequenzbereich wird auf nächsthöhere 2er-Potenz "gepadet" (kann allerdings eine Quelle für Ungenauigkeiten sein!), bei Rückkehr in Ortsbereich wird die ursprüngliche Bildgröße wieder ausgeschnitten
- Nachteil: nur unzureichende Dokumentation (wie man an die FFT-Ergebnisse herankommt, was Fast Hartley Transform ist, wie das Zusammenspiel zw. Plugin FFT und FHT ist usw.)
- Nachteil: die Tatsache, dass intern mit einer FHT gearbeitet wird, macht das Handling etwas unübersichtlicher. Ferner: Man muss sich den Zugriff auf bestimmte Windows besorgen, um an die FFT-Daten zu kommen!
- Nachteil: es ist unklar, wie die Windows letztlich heißen, auf die FFT-Plugin seine Ergebnisse ablegt (!!)

#### Vor- und Nachteile FFTJ Plugin

Download von von <https://imagej.nih.gov/ij/plugins>, macht 3D-FFT)

- Vorteil: ist für ein erstes Kennenlernen der FFT (z.B. im WPF) praktischer (WY-SIWYG)
- Vorteil: kann auch Stacks von Bildern (RealStack, ImagStack) transformieren (das ist die Bedeutung von "3D")
- Vorteil: besser dokumentiert, s. [plugins\FFTJ\FFTJ.html \(.txt\)](#) und [plugins\FFTJ\DeconvolutionJ.html \(.txt\)](#) (und es ist eben auch direkt eine Deconvolution damit verbunden)
- Vorteil: Geht auch für beliebige Bildgrößen (dies kann exakter sein!), aber deutliche Verlangsamung, wenn keine 2er-Potenz
- Nachteil: deutlich langsamer als FHT, wenn keine 2er-Potenz (aber fast gleich schnell bei 2er-Potenz)
- Nachteil: Java-Methoden nur erreichbar, wenn eigener Code im Ordner plugins/FFTJ liegt

## Built-In FFT und FHT im Detail

<https://imagejdocu.tudor.lu/gui/process/fft>

Die Klassen

- ij.plugin.FFT und
- ij.process.FHT

sind Teil der ImageJ-Distribution, sie werden über die Befehle in Menu Process – FFT angesprochen.

Die Philosophie beim Zusammenspiel von FFT und FHT ist die folgende: Für die eigentliche Arbeit wird ein FHT-Objekt (Fast Hartley Transform) erzeugt, das aber nach außen nicht direkt in Erscheinung tritt. Ein Objekt der Klasse FHT ist ein FloatProcessor. Es enthält die notwendige Information einer FFT im Frequenzbereich, allerdings anders angeordnet als bei einer FFT. Es wird wie folgt gespeichert:

- Entweder man wählt bei der FFT die Option "Display: FHT", dann wird der FloatProcessor der FHT als ImagePlus-Objekt mit Titel "FHT of ..." angezeigt, auf ihm kann eine inverse FFT ausgeführt werden.
- Oder man wählt bei der FFT die Option "Display: FFT", dann wird ein ImagePlus-Objekt mit Titel "FFT of ..." angezeigt, wobei die Anzeige allerdings nur ein 8bit-Grauwertbild ist, das das logarithmierte Powerspektrum enthält. Wo steckt die vollständige Information der FFT bzw. FHT? – Sie ist als Property vom Typ "FHT" an das Powerspektrumbild "drangeklebt" und kann mit `FHT fht = (FHT)imp.getProperty("FHT");` abgerufen werden.
- Oder man wählt bei der FFT die Option "Display: Complex", dann wird ein ImageStack mit Titel "Complex of ..." angezeigt, der Real- und Imaginärteil der FFT als 32bit-Floats enthält.

Ein FHT-Objekt kann mit Konstruktor `FHT(ImageProcessor ip)` von ImageProcessoren beliebigen Typs erzeugt werden. Dabei muss aber ip die Größe  $2^N \times 2^N$  haben. Da dies in der Regel nicht für jedes Bild zutrifft, übernimmt die Klasse FFT in den Methoden `newFHT(ip)` und `pad(ip)` den Job, das Bild fallweise auf die nächstgrößere Zweierpotenz zu "padding", wobei mit dem mittleren Grauwert des Bildes aufgefüllt wird.

Wie kann man eine Fourier-Transform von beliebigen Bildtypen machen?

- Bei ByteProcessor, ShortProcessor wird auf FloatProcessor konvertiert.
- Bei ColorProcessor werden die Fourieroperationen auf der extrahierten und nach FloatProcessor konvertierten **Brightness** durchgeführt. (Das gesamte RGB-Bild wird in `fht.rgb` zwischengespeichert, nachher bei einer inversen Transformation dieser FHT wird die neue Brightness wieder in `fht.rgb` eingefügt). Auf diese Weise lässt sich auch eine FFT von **Farb** Bildern durchführen!

Beim Padding merkt sich die FFT die ursprüngliche Bildgröße. Wird später von einem FHT-Objekt wieder eine inverse Transformation gemacht, so leistet `FFT.unpad(ImageStack)` oder

aber das crop() in FFT.doInverseTransform(fht) den Job, wieder das Bild in Originalgröße zu restaurieren.

Mögliche Aufrufe von **fft.run(String arg)**, wenn fft ein FFT-Objekt ist, diese werden alternativ aktiviert durch entsprechenden [Befehl] in SubMenu Process – FFT:

arg [Befehl]	aktuelles Image- Plus imp mit Titel imTit	Aufgerufene Methode Wirkung
"options" [FFT options..]	*	<b>FFT.showDialog()</b> Dialog zur <a href="#">Display-Einstellung</a> ; falls "Do Forward Transform" weiter bei arg="fft"
"redisplay" [Redisplay power spect- rum]	*, aber mit FHT- Property	<b>FFT.redisplayPowerspectrum()</b> erneute Anzeige des Frequenzspektrums, abhängig von den aktuellen <a href="#">Display-Einstellungen</a>
"swap" [Swap Quad- rants]	*, aber mit FHT- Property	<b>FFT.swapQuadrants(imp.getStack())</b> (not public!) Vertauschen der Quadranten, also Wechsel des Ursprungs zw. li. obere Ecke und Mitte
"inverse" [Inverse FFT]	Stack mit 2 Bildern (Real- und Imagi- när-Teil)	<b>FFT.doComplexInverseTransform()</b> (not public!) es wird angenommen, dass die beiden $2^N \times 2^N$ - Bilder eine komplexe Fourier-Trafo enthalten, diese werden in Ortsraum zurücktransformiert, evtl. "unpad" u. angezeigt
	Titel beginnt mit "FHT of"	<b>FFT.doFHTInverseTransform()</b> (not public!) es wird angenommen, dass imp eine $2^N \times 2^N$ - FHT (32bit) enthält, diese wird in den Orts- raum zurücktransformiert
* (auch "fft") [FFT]	*, ohne FHT-Pro- perty	<b>FHT fht = newFHT(ip); FFT.doFor- wardTransform(fht)</b> neue FHT erzeugen (fallweise nur von der ROI in imp, falls ROI definiert), diese in Frequenz- raum transformieren. Anzeige abhängig von aktuellen <a href="#">Display-Einstellungen</a> .
	*, mit FHT-Property	<b>FFT.doInverseTransform(fht)</b> (not public!) FHT aus imp-Objekt "abholen" und diese in- vers transformieren in Ortsraum

(\* bedeutet: beliebiger Wert für arg / beliebiger Typ für imp)

Es ist leider nicht so richtig klar, wie bei **fft.run("inverse")**; das Ergebnisbild nun heißt.

Display-Einstellungen (betreffen die Darstellung im Frequenzbereich):

Checkbox in Dialog "FFT options... "	public static Variable in Klasse FFT	Wirkung
Display: FFT	displayFFT (nicht public, aber normalerweise =true)	nach doForwardTransform wird log(PowerSpektrum) angezeigt, hat FHT-Property Window-Titel: <b>"FFT of " + imTit</b>
Display: Raw Power Spec	displayRawPS	nur unlogarithmiertes PowerSpekt- rum

Display: FHT	displayFHT	Anzeige der 32bit-FHT Window-Titel: <b>"FHT of " + imTit</b>
Display: Complex Stack	displayComplex	Anzeige von Real- und Imaginärteil der FFT in einem 2-Ebenen-Stack Window-Titel: <b>"Complex of " + imTit</b>

Normalerweise sind die Daten im Frequenzraum so abgelegt, dass DC-Anteil (Null) **AT\_CENTER**. Mit der Methode `FFT.swapQuadrants(ImageStack)`, die man über `fft.run("swap");` aktivieren kann, wenn `fft` ein FFT-Objekt ist, kann man die Frequenz-Null nach `AT_ZERO` verschieben

Man kommt an die Daten heran, indem man das `ImagePlus`-Objekt aus dem entsprechenden Window holt, z.B.

```
ImagePlus imp2 = WindowManager.getImage("Complex of "+imTitle);
```

dies liefert in `imp2` einen Stack (Realteil, Imaginärteil) zurück.

Beispiel [<ImageJ>\plugins\WPF\\_BV\Jet\\_Frequency.java](#): Wie kann ich im Bild `Jet-frequency.jpg` die Wellenstörung unterdrücken? Hierzu muss man im Frequenzbereich geeignet modifizieren. Auszug aus dem Plugin-Code (`ImagePlus imp` enthält das Bild `Jet-frequency-8bit.jpg`):

```
String imTitle = imp.getTitle();
int orig_w=imp.getWidth();
int orig_h=imp.getHeight();
FFT.displayComplex=true;
FFT.displayFHT=false;
FFT fft = new FFT();
(1)  fft.run("fft");          // Hin-FFT

(2)  ImagePlus imp2 = WindowManager.getImage("Complex of "+imTitle);

    // Hier kommt der Code um den Frequenzbereich zu modifizieren.
    // Setze bestimmte Zeilen (xa[i],*) und Spalten (*,ya[i]) komplett
    // auf val=0:
    ImageStack stack2 = imp2.getStack();
    int iw=imp2.getWidth();          // Bild ist quadratisch
    int val=0;
    int i,k;
    int[] xa = {110,111,145,146};    // abgelesen aus Power-Spektrum
    int[] ya = {102,103,153,154};    // für Bild Jet-frequency-8bit.jpg

    for (int s=1; s<=2; ++s) {
        ImageProcessor ip2 = stack2.getProcessor(s);
        for (k=0; k<xa.length; ++k) {
            for (i=0; i<iw; ++i) ip2.putPixel(xa[k],i,val);
            for (i=0; i<iw; ++i) ip2.putPixel(i,ya[k],val);
        }
    }
    imp2.setSlice(2);              // damit wird die neue Info im Stack
    imp2.setSlice(1);              // in beiden Slices auch angezeigt

(3)  WindowManager.setTempCurrentImage(imp2);
(4)  fft.run("inverse");          // Rück-FFT, Ergebnis ist im Stack mit
                                     // Namen imTitle+"-1"
```

In Zeile (1) passiert die Hin-FFT mit dem aktuellen Bild `imp`, basierend auf den vorher gemachten Display-Einstellungen.

Das Ergebnis, ein  $2^N \times 2^N$  großer 2-lagiger Stack, müssen wir uns in Zeile (2) mit dem `WindowManager` abholen.

Danach folgt die Modifikation im Frequenzbereich: bestimmte, vorher manuell ermittelte Zeilen und Spalten werden einfach auf val=0 gesetzt.  
In Zeile (3) machen wir den modifizierten Stack imp2 zum aktuellen Image.  
In Zeile (4) rufen wir damit die Rück-FFT. Das Ergebnis ist ein Stack, wieder mit der Originalgröße von Bild imp.

## FFTJ Plugin im Detail

Dieses Plugin muss man von <https://imagej.nih.gov/ij/plugins> herunterladen und im Plugins-Ordner installieren.

Kurzbeschreibung zur Nutzung:

- [plugins\FFTJ\FFTJ.html \(.txt\)](#) und
- [plugins\FFTJ\DeconvolutionJ.html \(.txt\)](#)

Die Philosophie bei FFTJ ist etwas anders: Wenn eine Dimension im Bild keine 2er-Potenz ist, wird eine (sehr langsame) Diskrete Fourier-Transformation (DFT) gemacht. Nur bei 2er-Potenzen wird die schnelle FFT angeworfen. Will man also die Schnelligkeit nutzen, so muss man vorher das Bild auf 2er-Potenz paden (und nachher auch wieder richtig ausschneiden, eine etwas lästige Arbeit, die einem ij.plugin.FFT abnimmt).

Dafür hat man bei FFTJ besseren Zugriff auf die Methoden und die FFT-Daten.

Allerdings hat das Schwester-Plugin DeconvolutionJ verschiedene Resize-Optionen eingebaut (noch zu checken).

## API-Kurz-Doku zu FFTJ

Wichtige Klassen in Ordner fftj sind

Klasse	statische Konstanten
fftj.FourierDomainOrigin	AT_ZERO AT_CENTER
fftj.ComplexValueType	ABS IMAG_PART REAL_PART FREQUENCY_SPECTRUM FREQUENCY_SPECTRUM_LOG PHASE_SPECTRUM POWER_SPECTRUM POWER_SPECTRUM_LOG
fftj.FFT3D fftj.SinglePrecFFT3D	die Hauptklassen

Die Hauptarbeit wird in den Transformerklassen FFT3D bzw. SinglePrecFFT3D getan, die sich nur dadurch unterscheiden, dass sie mit Double- oder Single-precision rechnen.

Die wichtigsten Methoden von FFT3D bzw. SinglePrecFFT3D sind

FFT3D(ImageStack sourceReal, ImageStack sourceImag)	Konstruktor, wird befüllt mit Ausgangsdaten, die auch Stacks enthalten können (3D) (sourceImag darf auch fehlen, = null)
void fft()	Vorwärts-Fouriertrafo
void ifft()	Inverse Fouriertrafo
ImagePlus toImagePlus(fftj.ComplexValueType type, fftj.FourierDomainOrigin fdOrig)	Hole aus FFT3D die durch ComplexValueType angeforderten Daten. fdOrig darf auch fehlen, dann ist AT_ZERO der Default

Wenn die Ausgangsdaten Stacks sind (gleiche Größe erforderlich), wird die FFT separat für jeden Slice der Stacks durchgeführt.

**ACHTUNG:** Wenn man mit den Daten weiterrechnen will bzw. einen neuen FFT3D-Transformer damit befüllen will, dann ist nur **AT\_ZERO** die richtige Wahl!! (Benutzt man dagegen AT\_CENTER zum Extrahieren von Real- und Imaginärteil im Frequenzraum, dann führt das Befüllen eines neuen Transformers mit genau diesen identischen Frequenzteilen bei anschließender Rück-FT zu einem Ortsraum-Bild, das rasterartig aussieht (jedes zweite Pixel in konstanten Bereichen ist das negative seines Vorgängers) (!!)) Das liegt daran, dass FFT3D die Frequenzen falsch interpretiert, sie erwartet die Null-Frequenz bei Array-Element [0][0].

### Beispiel "Idealer" Tiefpass

Beispiel [<ImageJ>\plugins\FFTJ\Ideal\\_LP\\_FFTJ.java](#): Wie kann ich auf ein Bild (z.B. building-256x256.jpg) einen "idealen" Tiefpassfilter anwenden? Dient gleichzeitig als Demo, wieso der "idele" Tiefpass überhaupt nicht ideal ist, sondern auf einem Bild "Ringing-Artefakte" produziert:

```
String imTitle = imp.getTitle();

(1)  fftj.FourierDomainOrigin fdOrigin
      = fftj.FourierDomainOrigin.AT_ZERO;
      fftj.ComplexValueType type
          = fftj.ComplexValueType.FREQUENCY_SPECTRUM_LOG;

IJ.showStatus( "Calculating Fourier Transform ..." );
(2)  fftj.FFT3D transf = new fftj.SinglePrecFFT3D(imp.getStack(),null);
      transf.fft();

// show log(frequency spectrum) with origin at image center
ImagePlus imp2 = transf.toImagePlus(type,
                                     fftj.FourierDomainOrigin.AT_CENTER);
imp2.show();

// here comes the code to modify content in the frequency domain:
(3)  for (int radius=25; radius<100; radius+=25) {
      // radius: cut off all wave numbers larger than radius
(4)  ImagePlus real = transf.toImagePlus(
          fftj.ComplexValueType.REAL_PART,fdOrigin);
      ImagePlus imag = transf.toImagePlus(
          fftj.ComplexValueType.IMAG_PART,fdOrigin);
      int iw=real.getWidth();
      int ih=real.getHeight();
      int val=0;
      for (int k=-iw/2; k<iw/2; ++k) {
          for (int i=-ih/2; i<ih/2; ++i) {
              if (k*k+i*i>radius*radius) {
                  real.getProcessor().putPixel((k+iw)%iw, (i+ih)%ih, val);
                  imag.getProcessor().putPixel((k+iw)%iw, (i+ih)%ih, val);
              }
          }
      }

      IJ.showStatus( "Calculating Inverse Fourier Transform ..." );
(5)  fftj.FFT3D itransf = new fftj.SinglePrecFFT3D(
          real.getStack(), imag.getStack());

      itransf.ifft();
      ImagePlus imp3 = itransf.toImagePlus(
          fftj.ComplexValueType.REAL_PART);
      imp3.setTitle("Ideal LP"+radius+" for "+imTitle);
      imp3.show();
  } // radius
```

- (1) Die statischen Klassenkonstanten müssen mit "fftj." angesprochen werden, damit sie gefunden werden.
- (2) Auch wenn es, wie hier, nur ein einzelnes Bild ist, muss es mit `imp.getStack()` an `transf` übergeben werden.
- (3) Wir rechnen aus einer Hin-FFT drei verschiedene Rück-FFTs um die Ringing-Artefakte für 3 verschiedene Tiefpässe zu zeigen.
- (4) Hier werden Real- und Imaginär-Frequenzteil aus dem `transf` geholt und nachfolgend der Frequenzbearbeitung unterworfen.
- (5) Hier wird ein neuer `itransf` mit den Frequenzdaten befüllt, nachfolgend die Rück-FFT aufgerufen und der Realteil (der das veränderte Bild enthält) extrahiert und angezeigt.  
**NOCHMAL:** Wenn `itransf` den Real- und Imaginärteil für die Frequenzdomäne bekommt, erwartet er die Nullfrequenz (DC-Anteil) `AT_ZERO`.

## ***Fazit***

Das Wrapping in der Klasse `ij.plugin.FFT` (ImageJ-Built-In) scheint softwaretechnisch nicht so ganz gelungen zu sein (es ist etwas kompliziert, die FHT richtig zu befüllen, und es ist auch nicht immer so ganz klar, wo die Ergebnisdaten abgelegt werden). Wenn man also die FHT im eigenen Java-Code nutzen will, empfiehlt es sich vielleicht, seinen eigenen Java-Wrapper zu schreiben.

Für erste Plugin-Experimente im Fourierraum empfiehlt sich das einfacher zu durchschauende `FFTJ`.