Fast Fourier Transform - Improvement, Variants and Applications

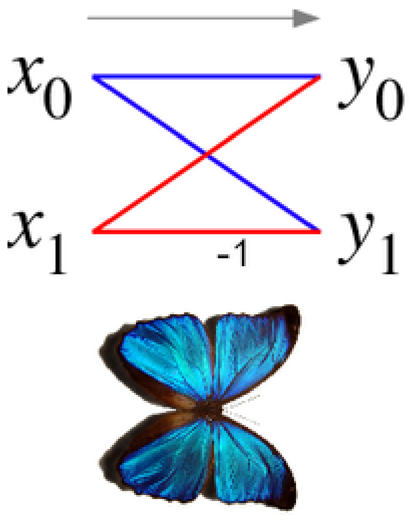
Name: 陈康睿

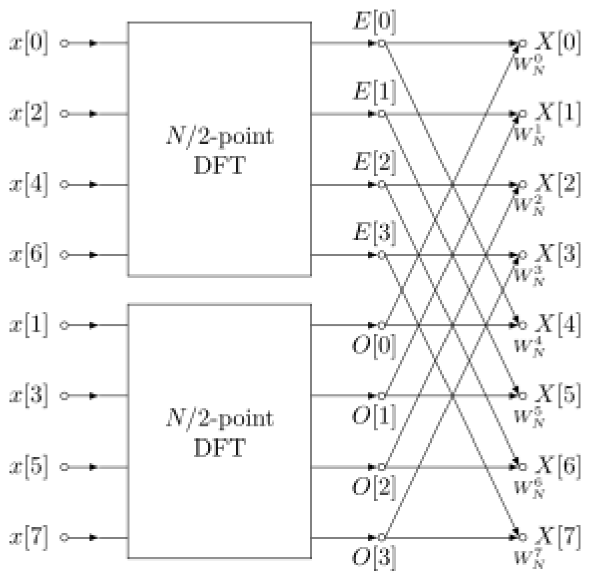
SID: 12110524

Butterfly Transform

In our lecture, FFT algorithm ends with its recursive implementation, which results in large constant-level time as well as huge memory cost. So we could manage to find a way to improve it by use iteration via doubling.

This optimization is called "Butterfly Transform" because its process looks like a butterfly.





By observing, we could easily find that on the deepest level, the index where an original number stands is the reverse of its original binary bits. So we could rearrange the array in $O(n)$ time and double the merge each iteration to simulate the divide & conquer. By doing so, though the time complexity stays the same, the algorithm runs much faster than before, and the additional memory cost is reduced to $O(1)$.

```cpp
// C++ implementation
inline void change(Complex_d *a, int n)
{
    for (int i = 0; i < n; ++i)
    {
        rev[i] = rev[i >> 1] >> 1;
        if (i&1)
            rev[i] |= n >> 1;
    }
    for (int i = 0; i < n; ++i)
        if (i < rev[i])
            swap(a[i], a[rev[i]]);
    return;
}

inline void fft(Complex_d *a, int n, int flag)
{
    change(a, n);
    for (int h = 2; h <= n; h <<= 1)
    {
        Complex_d wn(cos(2*PI/h), sin(flag*2*PI/h));
        for (int j = 0; j < n; j += h)
        {
            Complex_d w(1, 0);
            for (int k = j; k < j + h / 2; k++)
            {
                Complex_d u = a[k];
```

```
                    Complex_d t = w*a[k+h/2];
                    a[k] = u+t;
                    a[k+h/2] = u-t;
                    w = w*wn;
                }
            }
        }
        if (flag == -1)
            for (int i = 0; i < n; i++)
                a[i].real(a[i].real()/n);
}
```

Number Theoretic Transform

In Discrete Mathematics, we have learnt the concept of **primitive root**,

> A primitive root mod n is an integer g such that every integer relatively prime to n is congruent to a power of g mod n. ,That is, the integer g is a primitive root $mod\ n$ if for every number a relatively prime to n there is an integer z such that $a \equiv g^z\ mod\ n$.

Here we got a property for a primitive root g is that $(g_n^{k+\frac{n}{2}})^2 = g_n^{2k+n} \equiv g_n^{2k}\ (mod\ p)$, which is quite similar to the relation between w_n^k and $w_n^{k+\frac{n}{2}}$ for a unit root w. So like using **DFT** to transform real number field to complex number field, here we could use **NTT** to transform integer number field to finite field (as the primitive root is usually small, so we could just use enumeration to find it). And the idea of divide & conquer of **FDFT** can also be imply here, so we got another **FFT** algorithm - **FNTT**.

```
// C++ implementation
void ntt(int *x, int lim, int opt)
{
    int i, j, k, m, gn, g, tmp;
    for (i = 0; i < lim; i++)
        if (r[i] < i)
            swap(x[i], x[r[i]]);
    for (m = 2; m <= lim; m <<= 1)
    {
        k = m >> 1;
        gn = qpow(3, (P-1)/m);
        for (i = 0; i < lim; i += m)
        {
            g = 1;
            for (j = 0; j < k; j++, g = g*gn%P)
            {
                tmp = x[i+j+k]*g%P;
                x[i+j+k] = (x[i+j]-tmp+P)%P;
                x[i+j] = (x[i+j]+tmp) % P;
            }
        }
    }
    if (opt == -1) {
        reverse(x+1, x+lim);
        int inv = qpow(lim, P-2);
        for (i = 0; i < lim; i++) x[i] = x[i]*inv%P;
    }
}
```

Though a bit faster than **FDFT**, here is a restriction for **FNTT**: the coefficients must be integers (of course because this comes from number theory).

Specific types of Matrix Multiplication Accelaration

Circulant

$$C = \begin{pmatrix} c_0 & c_{n-1} & \cdots & c_1 \\ c_1 & c_0 & \cdots & c_2 \\ \vdots & \vdots & \ddots & \vdots \\ c_{n-1} & c_{n-2} & \cdots & c_0 \end{pmatrix}$$

A circulant matrix can be diagonalized by the **DFT** matrix (the matrix formed by the powers of unit root in FFT):

$$C = F^{-1}\Lambda F$$

where F is the $n \times n$ **DFT** matrix and Λ is a diagonal matrix such that $\Lambda = diag(F\underline{c})$. Therefore a circulant matrix can be applied to a **vector** in $O(nlogn)$ operations using **FFT**.

Toeplitz

$$T = \begin{pmatrix} T_1 & \overline{T}_2 & \cdots & \overline{T}_{n-1} \\ T_2 & T_1 & \cdots & \overline{T}_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ T_n & T_{n-1} & \cdots & T_1 \end{pmatrix} = \begin{pmatrix} T_1 & 0 & \cdots & 0 \\ T_2 & T_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ T_n & T_{n-1} & \cdots & T_1 \end{pmatrix} + \begin{pmatrix} 0 & \overline{T}_2 & \cdots & \overline{T}_{n-1} \\ 0 & 0 & \cdots & \overline{T}_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

Now we can simply consider T to be a **triangular Toeplitz matrix**.

$$y = Tx$$

$$\begin{pmatrix} y \\ 0 \end{pmatrix} = \begin{pmatrix} T & \overline{T} \\ \overline{T} & T \end{pmatrix} \begin{pmatrix} x \\ 0 \end{pmatrix}$$

Here

$$\begin{pmatrix} T & \overline{T} \\ \overline{T} & T \end{pmatrix}$$

is a circulant matrix, so we can use the same technique from **circulant** matrix.

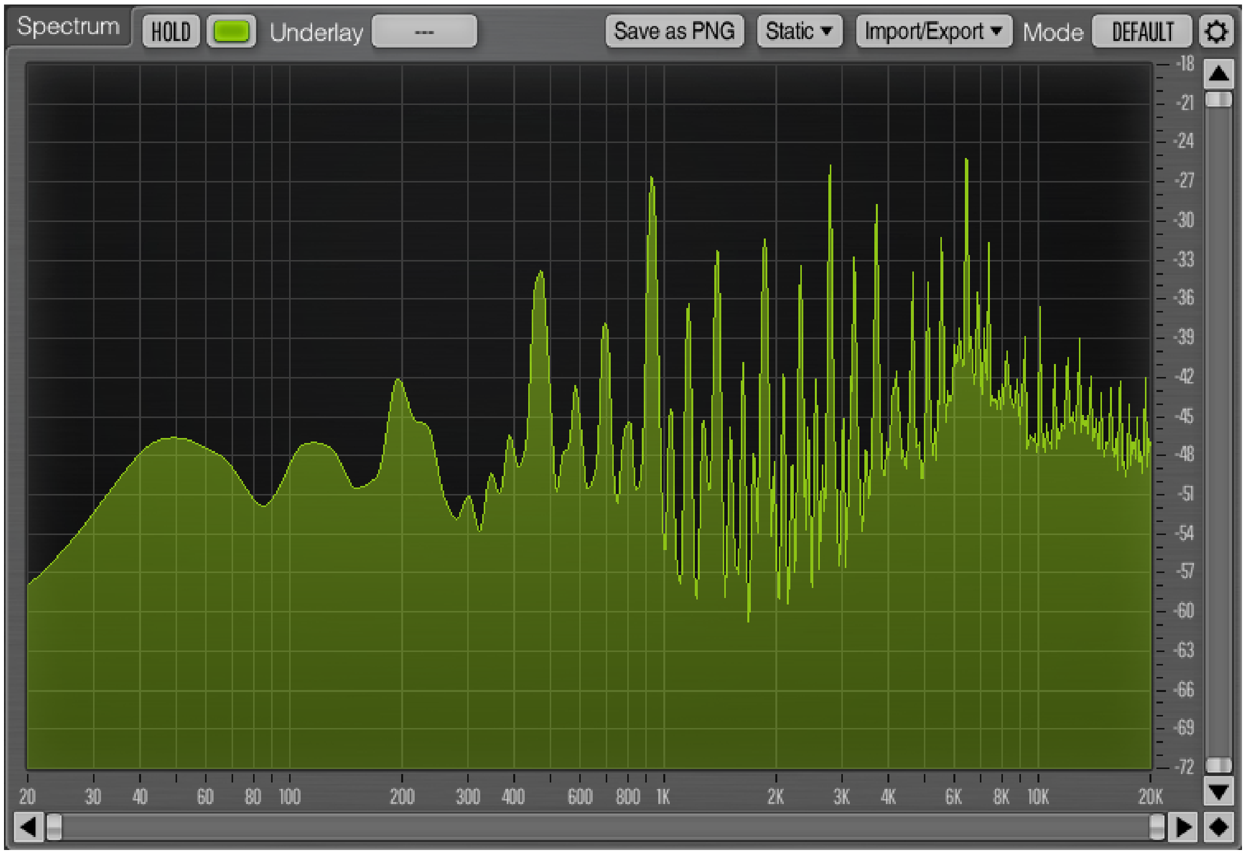
(Noticed that two **Toeplitz matrices** multipling together can represent a convolution.)

Hankel

Hankel matrix is the permutation of **Toeplitz matrix**, so after doing permutation transformation , we can simply use the same idea above to accelerate **Hankel matrx** applying to a **vector** via **FFT**.

FFT in Signal Process (Audio as an Example)

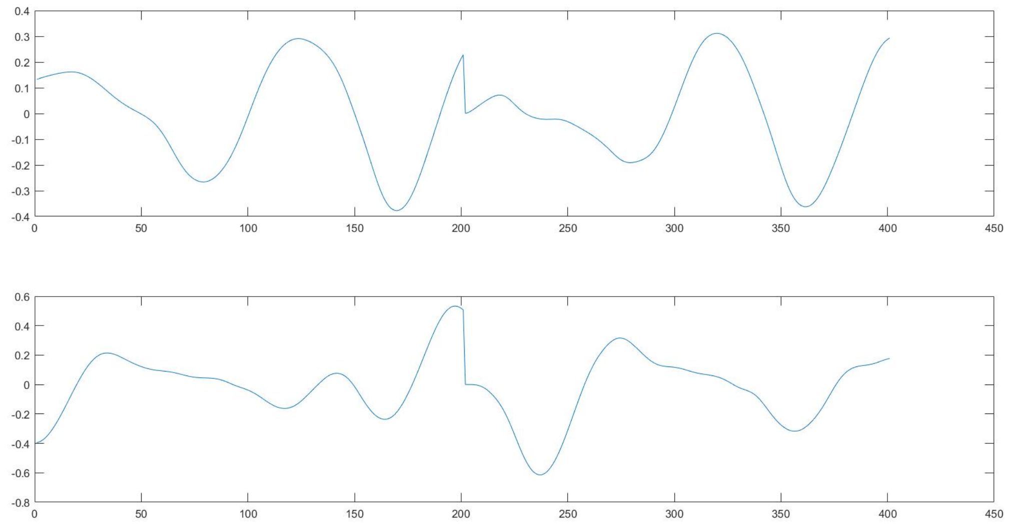
Recap the original usage of **Fourier Transform** - converting a function into a form that describes the frequencies (i.e. transforming between time domain and frequency domain). And the way to achieve that is to break the whole function down to a series of sin or cos functions. Intuitivly, it will play a important role in signal process. Here we take audio as an example. Analyzing audio from time-amplitude sampled wave data flow to frequency infomation helps designers, producers, mixers and master engineers to post-process audio works.



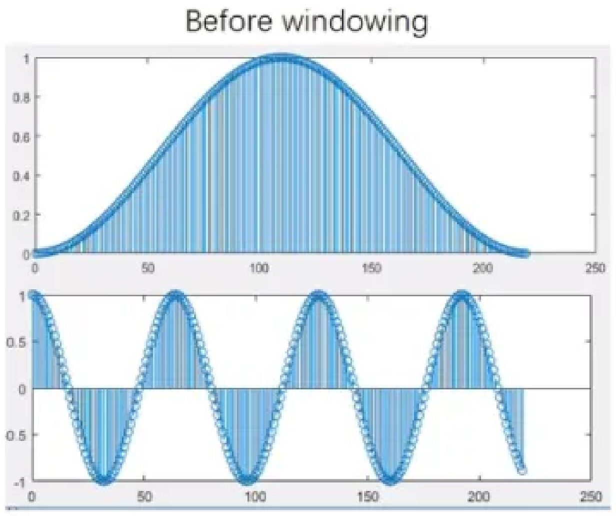
$$F(\omega) = \mathcal{F}[f(t)] = \int_{-\infty}^{\infty} f(t)e^{-iwt}\,dt$$

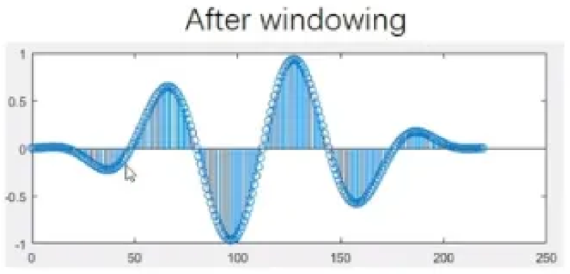
By understanding this formula, we could easily know the frequency spectrum we get is symmetric by a certain frequency. So there is no wonder why the standard sample rate is at least 44.1kHz (just a bit larger than 2 times of 20kHz - the highest frequency that most of us humen beings can hear).

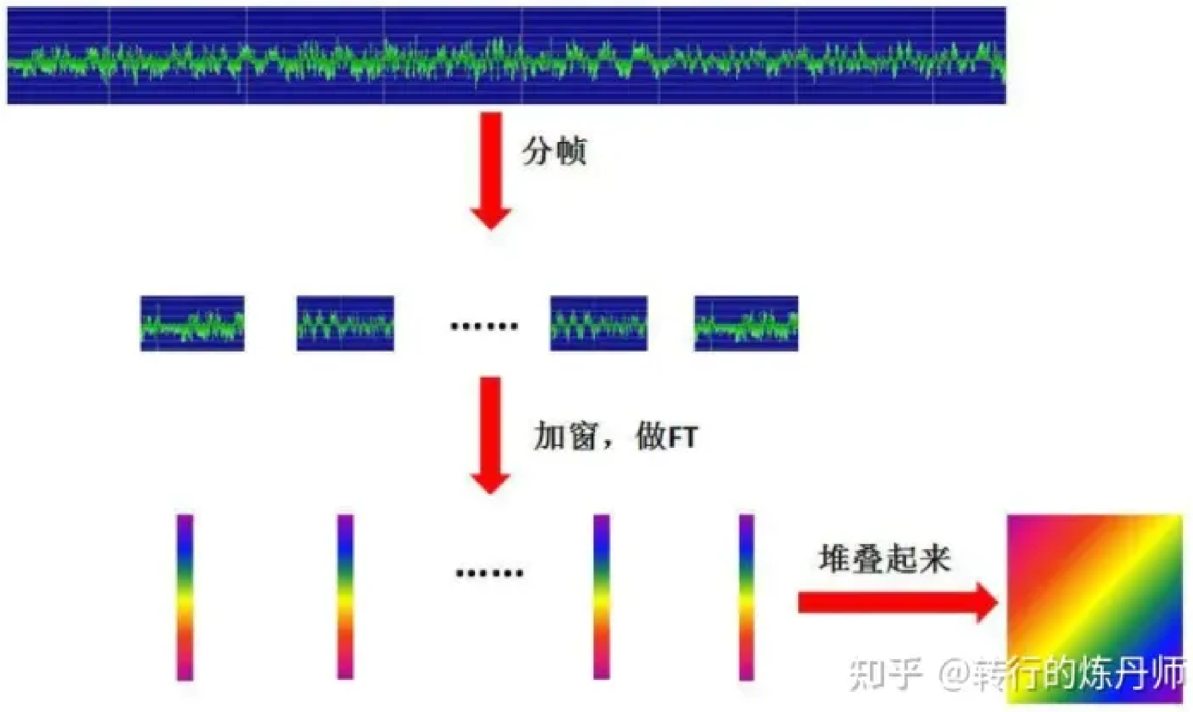
However, **Fourier Transform** itself doesn't carry time information when converting between different domains, which will lead to some unexpected results. For instance, if we chop out a piece of audio, modify its frequency information by **FFT** and **RFFT** and then push it back. We are supposed to find that though it sounds right individually, its phase has been changed and thus can not match the other parts. This will cause a sudden high frequency flash and sound quite noisy.



To conquer this problem, scientists invented several techniques and algorithms. One of them is **Short-time Fourier Transform**. As audio samples are discrete and thus have error, the request can be eased to "smoothing it as much as we can not hear it" rather than idealy recovery the percise time and phase infomation. Here are two key techniques to achieve this target: **framing** and **windowing**. Framing: chop the audio into small pieces; windowing: use a function to compress the amplitude. Then merge the frames after processing. By adjusting parameters like frame size, hop size etc. accordingly, we can get more accurate infomation.







Other approach like **Continue Wavelet Transform** will not be cover in this report.

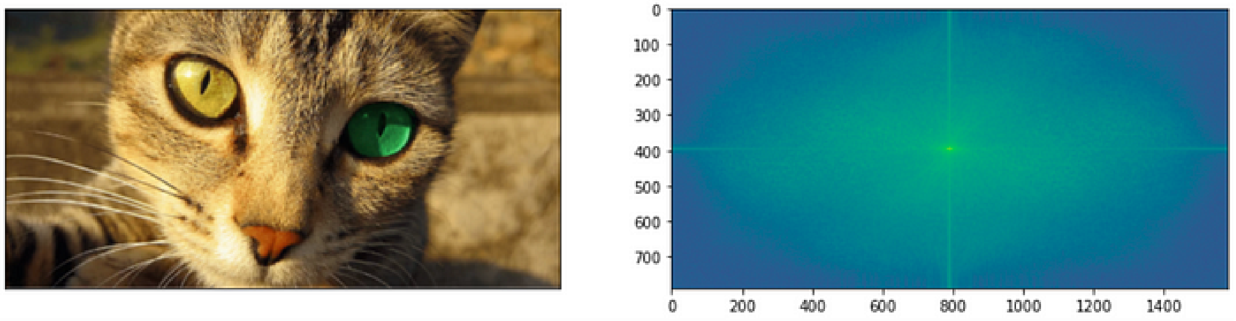
Multidimensional FFT

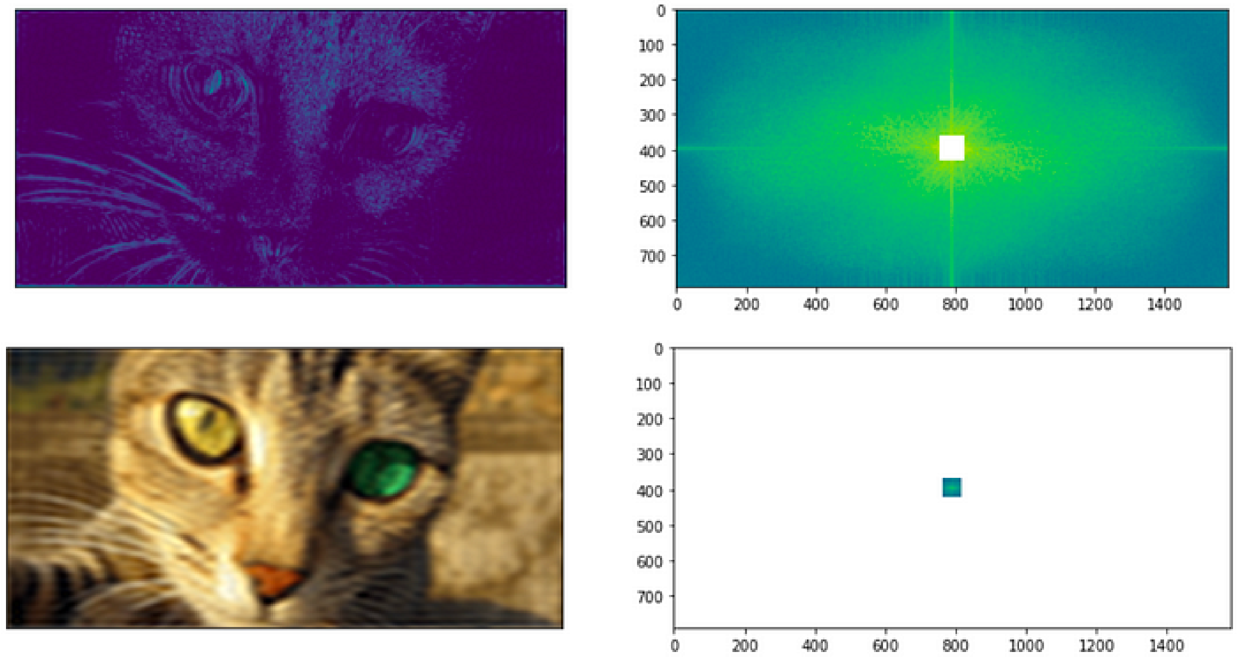
Fourier Transform on higher dimensions become more complex because of multidimensianal integration.

2D (Image as Example)

$$F(u,v)=\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}f(x,y)e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$

In a 2D image, **Fourier Transform** can transform the spatial domain to frequency fomain. As for image, spatial domain contains the grey scale information, and frequency domain indicates whole (lower frequency) and detail (higher frequency).

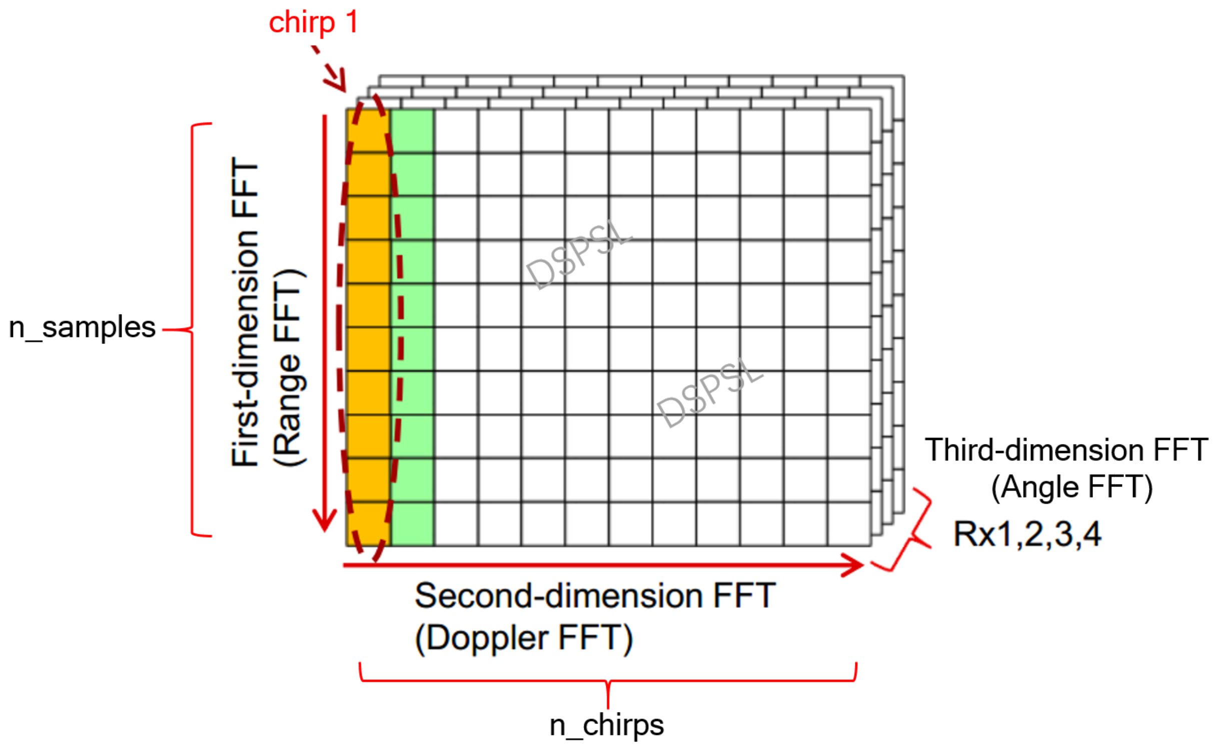




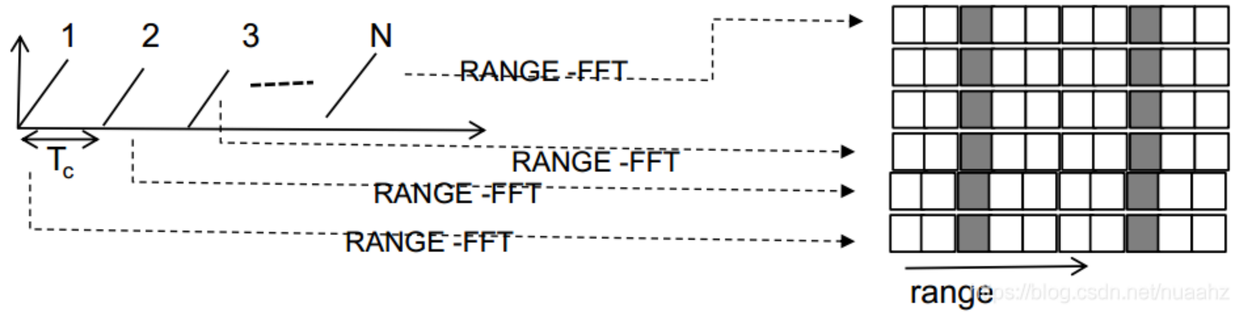
By modifying frequency spectrum, we are able to wipe some regular noise, also we could choose to reduce some high frequencies to compress data, making trade-off between details/clearity and file size.

3D (Radar as Example)

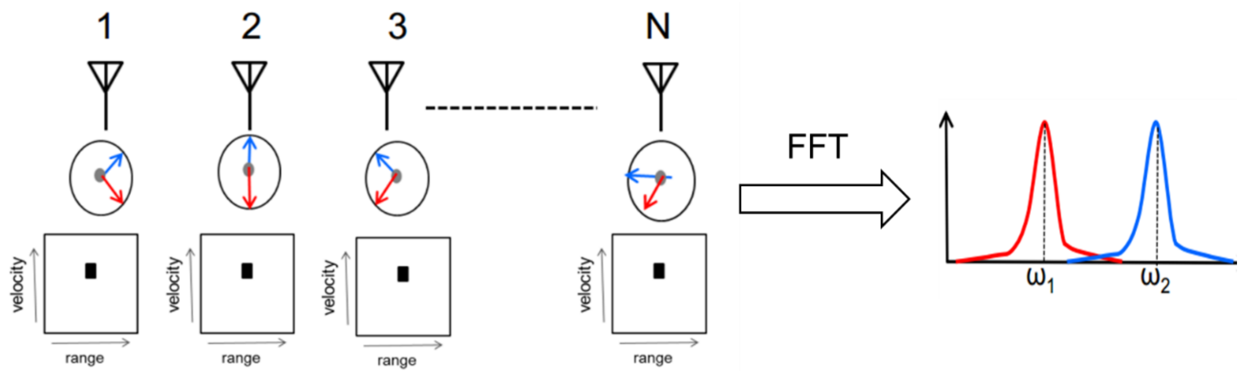
In **LFMCW Radar System**, **Fourier Transform** is used to transfer distance (delay), velocity (dropplet), angle (array) information to frequency domain (which indicates the change rate of these three dimensions).



chirp 1
n_samples
First-dimension FFT (Range FFT)
DSPSL
DSPSL
Third-dimension FFT (Angle FFT)
Rx1,2,3,4
Second-dimension FFT (Doppler FFT)
n_chirps



1
2
3
N
RANGE -FFT
RANGE -FFT
RANGE -FFT
RANGE -FFT
T_c
range
https://blog.csdn.net/nuaahz



1
2
3
N
velocity
range
FFT
ω_1
ω_2