Generalized FFTs: Survey and Applications

Hy Truong Son
Advisor: Risi Kondor
Topics in Statistical Machine Learning (CMSC 35425)

The University of Chicago

May 2018

THE UNIVERSITY OF CHICAGO

Reference

- David K. Maslen & Daniel N. Rockmore: Generalized FFTs - A survey of some recent results
- Daniel N. Rockmore: Some applications of generalized FFTs
- Risi Kondor: Group theoretical methods in machine learning (chapter 3)

THE UNIVERSITY OF CHICAGO

Topics

Today, we will discuss the following main topics:

- DFT/FFT 1D, 2D
- Cooley-Tukey (radix-2) algorithm
- Fast Fourier Transform on $\mathbb{Z}_n$
- Non-commutative FFTs
- Clausen's FFT for the symmetric group
- Applications of DFT/FFT in signal/image processing
- Fast Polynomial/Integer Multiplication

THE UNIVERSITY OF CHICAGO

My Fourier Transform Library

Source code:

`https://github.com/HyTruongSon/Fourier-Transform-Library`

Functionalities:

- Fourier Transform Library (MATLAB interface based on MEX/C++)
- DFT, FFT 1D
- DFT, FFT 2D
- DCT 2D
- JPEG (without lossless compression)
- Fast polynomial multiplication
- Fast integer multiplication
- Other things

All figures in this presentation can be **reproduced** from my library.

THE UNIVERSITY OF CHICAGO

History

Cooley and Tukey were interested in the analysis of time series. The particular application that Cooley and Tukey had in mind early in the 1960's was the analysis of seismic data. At this time, a nuclear test with the (then) USSR was under negotiation. The USSR was balking at the notion of site visits, so it was necessary that there be some way of remotely confirming compliance.

The prevailing idea was to surround the Soviet Union with many sensors in order to monitor seismic activity. Nuclear detonations could then be detected by particular structure in the Fourier transforms of the collection of time series.

THE UNIVERSITY OF CHICAGO

Yates - Fast interaction analysis for 2^k-factorial designs

One version of the abelian FFT is due to the statistician and design theorist Yates. To efficiently compute the interaction analysis for data from a 2^k-factorial design, Yates descibed an algorithm which is an FFT for the group $(\mathbb{Z}/2\mathbb{Z})^k$.

A 2^k-factorial design is the set of all k-tuples of signs $\{+1,-1\}^k$, which can be thought of as the vertices of the k-dimensional hypercube or the space of binary k-tuples.

It is a natural way to index the trials of an experiment which depends on k **factors**, each of which maybe a set at a **high** or **low** level.

THE UNIVERSITY OF CHICAGO

Yates - Fast interaction analysis for 2^k-factorial designs

Example: Average height of plants, denoted by α_{swf} for a given choice of sunlight (s), weed killer (w) and fertilizer (f). We are given a dataset for a 2^3-factorial design.

The **zeroth** order effect is the grand mean or total average height:

$$\mu_{gr} = \frac{1}{8} \sum_{(s,w,f) \in \{+,-\}^3} \alpha_{swf}$$

Consider the **first** order effects: the effect of one particular factor, all other factors being held equal. Consider the differences of the average yields at a high level of sunlight versus the average at a low level:

$$\mu_S = \frac{1}{4}(\alpha_{+--} + \alpha_{+-+} + \alpha_{++-} + \alpha_{+++}) - \frac{1}{4}(\alpha_{---} + \alpha_{--+} + \alpha_{-+-} + \alpha_{-++})$$

Yates - Fast interaction analysis for 2^k-factorial designs

The collection of first, second and third order effects can be coded up as the computation of a matrix-vector multiplication (the matrix of size $2^k \times 2^k$). The analysis is the same as computing the **projection** of the data vector onto an **orthogonal basis**.

Naively, we need $(2^k)^2$ operations to compute the analysis. Let H_k denote the matrix of the Fourier transform on $(\mathbb{Z}/2\mathbb{Z})^k$. For example:

$$H_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Yates - Fast interaction analysis for 2^k-factorial designs

Any character of $(\mathbb{Z}/2\mathbb{Z})^3$ maybe written as a tensor product of characters of the group $\mathbb{Z}/2\mathbb{Z}$:

$$H_3 = H_1 \otimes H_1 \otimes H_1$$

$$H_3 = [I_4 \otimes H_1] \cdot [I_2 \otimes H_1 \otimes I_2] \cdot [H_1 \otimes I_4]$$

where I_j denotes the $j \times j$ identify matrix.

This is a **sparse decomposition** of the matrix H_3, and the Fourier transform of α is computed by multiplying by each of these sparse matrices in turn. Remark: **Walsh-Hadamard** transform.

THE UNIVERSITY OF CHICAGO

Fourier analysis for finite groups - 1

A (complex) **matrix representation** of a finite group G is a map ρ from G into the group $d \times d$ invertible matrices with complex entries, $GL_d(\mathbb{C})$, such that:

$$\rho(st) = \rho(s)\rho(t) \qquad (\forall s, t \in G)$$

where $d = d_\rho$ is called the **degree** or **dimension** of the representation ρ, and $V = \mathbb{C}^d$ is called the **representation space** of ρ. The function $\rho_{ij}(s)$ defined by considering the i, j entry of $\rho(s)$ for each s is called a matrix coefficient.

THE UNIVERSITY OF CHICAGO

Fourier analysis for finite groups - 2

Two representations ρ_1 and ρ_2 are said to be equivalent if they differ only by a change of basis, i.e., if there exists an invertible matrix A such that $\rho_1(s) = A^{-1}\rho_2(s)A$ for all $s \in G$.

A subspace $W \subset V = \mathbb{C}^d$ is said to be G-invariant if for all $s \in G$, $\rho(s)W \subset W$. The representation ρ is said to be **irreducible** if $V = \mathbb{C}^d$ has no G invariant subspaces other than the trivial subspaces $\{0\}$ and V. Otherwise, ρ is said to be **reducible**.

Any representation is equivalent to the direct sum of irreducible representations, where the direct sum of two representations is the matrix direct sum of the representations.

THE UNIVERSITY OF CHICAGO

Fourier analysis for finite groups - 3

Let $\rho^{(1)},..,\rho^{(h)}$ be a complete set of inequivalent matrix representations for G with d_i equal to the degree of $\rho^{(i)}$. Then the corresponding collection of matrix coefficients $\{\rho^{(i)}_{jk} | 1 \leq i \leq h, 1 \leq j,k \leq d_i\}$ form an orthogonal basis for the $|G|$ dimensional vector space of complex valued functions on G, denoted $\mathbb{C}G$.

A Fourier transform for a finite group G is a **change of basis** from the basis of point mass or delta functions for $\mathbb{C}G$ to a **basis of irreducible matrix coefficients**.

Fourier analysis for finite groups - 4

Let G be a finite group and f be a complex valued function on G. Let ρ be a matrix representation of G. Then the Fourier transformation of f at ρ, denoted $\hat{f}(p)$ is the matrix sum:

$$\hat{f}(\rho) = \sum_{s \in G} f(s)\rho(s)$$

Similarly, define the Fourier transform of f at the matrix coefficient ρ_{jk}, denoted $\hat{f}(\rho_{jk})$ as the scalar sum:

$$\hat{f}(\rho_{jk}) = \sum_{s \in G} f(s)\rho_{jk}(s)$$

THE UNIVERSITY OF CHICAGO

Fourier analysis for finite groups - 5

Let $\mathscr{R}$ be a set of matrix representations of G. Then the Fourier transform of f on $\mathscr{R}$ is the set of $|\mathscr{R}|$ (matrix) Fourier transforms of f at the representations in $\mathscr{R}$, or equivalently, the $\sum_{\rho \in \mathscr{R}} d_\rho^2$ scalar quantities given by the Fourier transforms of f at the matrix coefficients of the representations in $\mathscr{R}$.
The Fourier inversion formula:

$$f(s) = \frac{1}{|G|} \sum_{\rho \in \mathscr{R}} d_\rho \, trace\big(\hat{f}(\rho)\rho(s^{-1})\big)$$

Fourier analysis for finite groups - 6

The **convolution** of f with h, denoted by $f * h$, is defined as:

$$f * h(x) = \sum_{y \in G} f(xy^{-1})\, h(y)$$

For finite groups and for any irreducible representation ρ of G:

$$\widehat{f * h}(\rho) = \hat{f}(\rho) \cdot \hat{h}(\rho)$$

THE UNIVERSITY OF CHICAGO

Fourier analysis for finite groups - 7

Let G be a finite group, and $\mathscr{R}$ any set of matrix representations of G. The **complexity of the Fourier transform** for the set $\mathscr{R}$, denoted $T_G(\mathscr{R})$, is defined to be the minimum number of operations needed to compute the Fourier transform of f on $\mathscr{R}$ via a straight-line program for an arbitrary complex-valued function f defined on G. Define the **complexity of the group** G to be:

$$\mathscr{C}(G) = \min_{\mathscr{R}}\{T_G(\mathscr{R})\}$$

where $\mathscr{R}$ varies over all complete sets of inequivalent irreducible matrix representations of G.

Fourier analysis for finite groups - 8

The complexity for a set of representations is equal to the complexity for the direct sum of the representations:

$$T_G(\mathscr{R}) = T_G(\oplus_{\rho \in \mathscr{R}} \rho)$$

Direct computation of any Fourier transform gives the upper and lower bounds

$$|G| - 1 \leq \mathscr{C}(G) \leq T_G(\mathscr{R}) \leq |G|^2$$

The **reduced complexity** is defined as:

$$t_G(\mathscr{R}) \triangleq T_G(\mathscr{R})/|G|$$

Continuous Fourier Transform - 1

Fourier transform:

$$\hat{f}(w) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x).e^{-ixw} dw$$

Inverse Fourier transform:

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{f}(w).e^{ixw} dw$$

$\mathscr{F}(f) = \hat{f}(w)$ is the Fourier transform of $f(x)$.

Continuous Fourier Transform - 2

Plancharel theorem:

- The Fourier transform preserves inner product, if $\hat{f}$ is the Fourier transform of f and $\hat{g}$ is the Fourier transform of g, then:

$$\langle \hat{f}(w), \hat{g}(w) \rangle = \langle f(x), g(x) \rangle$$

- $\|f(x)\|^2 = \|\hat{f}(w)\|^2$

Some other basic properties of Fourier Transform:

- $\mathscr{F}(f(x-c))(w) = e^{-iwc}.\hat{f}(w)$
- $\mathscr{F}(f(cx))(w) = \frac{1}{c}.\hat{f}(\frac{w}{c})$

THE UNIVERSITY OF CHICAGO

Transition from continuous to discrete

To relate the Fourier transform of a continuous function to the transform of a finite sequence, we should make the following assumptions:

- The signal is a **periodic** function on the real line.
- The signal is **band-limited**, i.e., the signal has a finite Fourier expansion.

$$f(t) = \sum_{k=-N}^{N} \hat{f}(k) e^{-2\pi ikt} \qquad (0 \le t < 1)$$

$$\hat{f}(k) = \int_0^1 f(t) e^{2\pi ikt} dt$$

Quadrature rule with bandwidth N:

$$\hat{f}(k) = \frac{1}{2N+1} \sum_{j=0}^{2N} f\left(\frac{j}{2N+1}\right) e^{2\pi ikj/(2N+1)}$$

THE UNIVERSITY OF CHICAGO

Discrete Fourier Transform 1D - 1

Fourier transform:

$$\hat{f}_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} f_n . e^{-i2\pi k \frac{n}{N}}$$

In the case that the original signal (f_n) is real:

$$\hat{f}_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} f_n \left[\cos\left(2\pi k \frac{n}{N}\right) - i \sin\left(2\pi k \frac{n}{N}\right) \right]$$

$$Re(\hat{f}_k) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} f_n . \cos\left(2\pi k \frac{n}{N}\right)$$

$$Im(\hat{f}_k) = -\frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} f_n . \sin\left(2\pi k \frac{n}{N}\right)$$

THE UNIVERSITY OF CHICAGO

Discrete Fourier Transform 1D - 2

Inverse Fourier transform:

$$f_n = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \hat{f}_k . e^{i2\pi k \frac{n}{N}}$$

Amplitude:

$$|\hat{f}_k| = \sqrt{Re(\hat{f}_k)^2 + Im(\hat{f}_k)^2}$$

Phase:

$$arg(\hat{f}_k) = atan2(Im(\hat{f}_k), Re(\hat{f}_k))$$

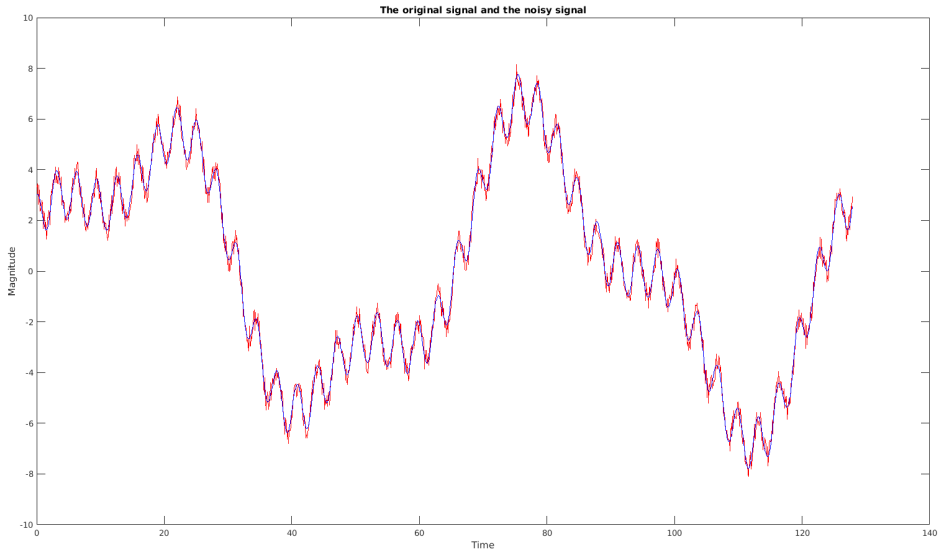
Source code:

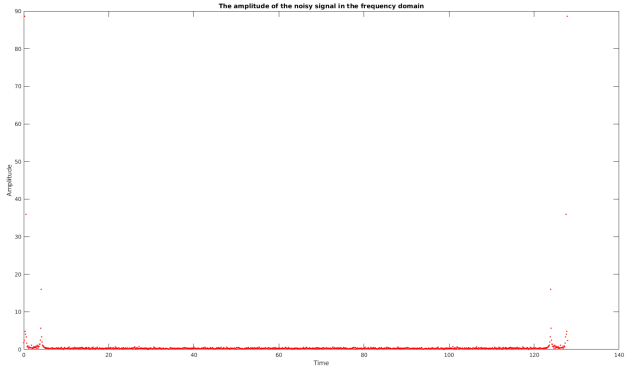
dft_1d.cpp

THE UNIVERSITY OF CHICAGO

Discrete Fourier Transform 1D - 3

example_dft_1d_denoising.m
example_fft_1d_denoising.m

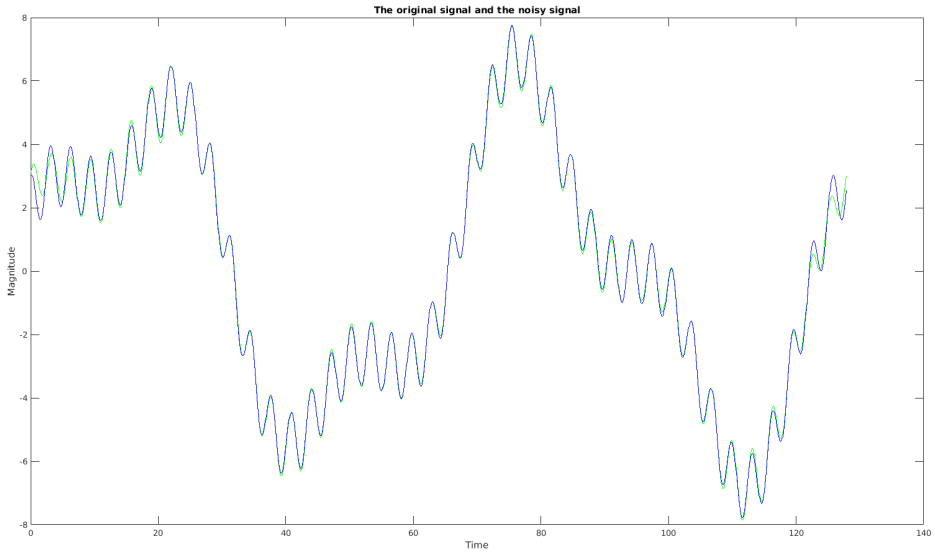


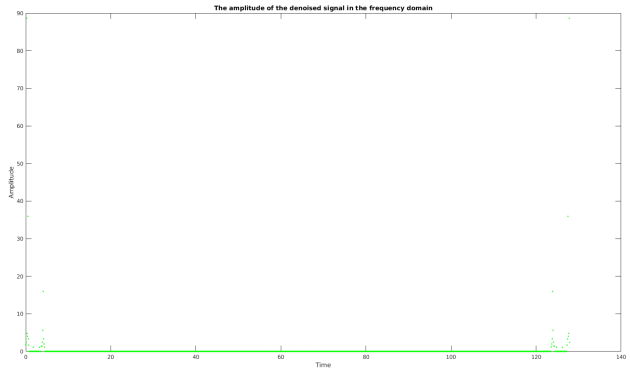


THE UNIVERSITY OF CHICAGO

Discrete Fourier Transform 1D - 4

example_dft_1d_denoising.m
example_fft_1d_denoising.m





THE UNIVERSITY OF CHICAGO

Discrete Fourier Transform 2D - 1

Fourier transform:

$$\hat{f}_{x,y} = \frac{1}{\sqrt{MN}} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} f_{u,v}.e^{-i2\pi(\frac{xu}{M}+\frac{yv}{N})}$$

$$\hat{f}_{x,y} = \frac{1}{\sqrt{MN}} \sum_{u=0}^{M-1} e^{-i2\pi\frac{xu}{M}}.\left(\sum_{v=0}^{N-1} f_{u,v}.e^{-i2\pi\frac{yv}{N}}\right)$$

Let $P(u,y) = \sum_{v=0}^{N-1} f_{u,v}.e^{-i2\pi\frac{yv}{N}}$. Then:

$$\hat{f}_{x,y} = \frac{1}{\sqrt{MN}} \sum_{u=0}^{M-1} e^{-i2\pi\frac{xu}{M}}.P(u,y)$$

THE UNIVERSITY OF CHICAGO

Discrete Fourier Transform 2D - 2

Inverse Fourier transform:

$$f_{u,v} = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \hat{f}_{x,y}.e^{i2\pi(\frac{xu}{M}+\frac{yv}{N})}$$

$$f_{u,v} = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} e^{i2\pi\frac{xu}{M}}.\left(\sum_{y=0}^{N-1} \hat{f}_{x,y}.e^{i2\pi\frac{yv}{N}}\right)$$

Let $H(x,v) = \sum_{y=0}^{N-1} \hat{f}_{x,y}.e^{i2\pi\frac{yv}{N}}$. Then:

$$f_{u,v} = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} e^{i2\pi\frac{xu}{M}}.H(x,v)$$

Source code:

dft_2d.cpp

THE UNIVERSITY OF CHICAGO

Discrete Fourier Transform 2D - 3

The Fourier image (to visualize amplitudes or phases) is shifted in such a way that the value (image mean) $\hat{f}(0,0)$ is displayed in the center of the image. The further way from the center an image point is, the higher is its corresponding frequency. The dynamic range of the Fourier coefficients (for example, the intensity values in the Fourier image) is too large to be displayed on the screen, therefore all other values appear as black. We need to apply a logarithmic transformation to the image as follows:

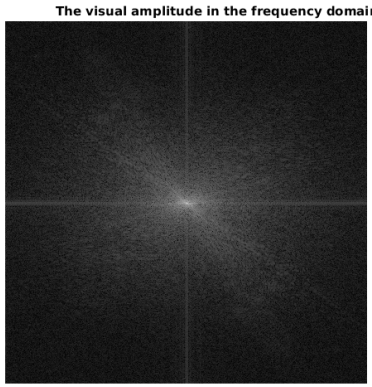
$$Q(i,j) = c\log(1 + |P(i,j)|)$$

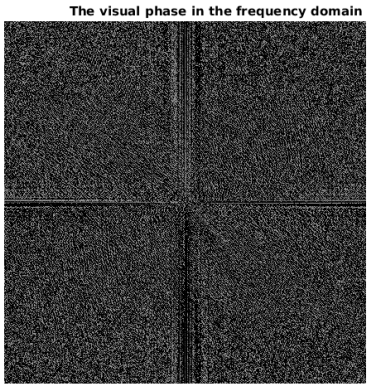
where $P(i,j)$ is the original image, and $Q(i,j)$ is the transformed image. The scaling constant c is chosen so that the maximum output value is 255 (providing an 8-bit format). That means if R is the value with the maximum magnitude in the input image, c is given by:

$$c = \frac{255}{\log(1 + |R|)}$$

Discrete Fourier Transform 2D - 4

example_dft_2d.m
example_fft_2d.m





THE UNIVERSITY OF CHICAGO

Fast Fourier Transform 1D (radix-2) - 1

Cooley-Tukey FFT algorithm was invented by Carl Friedrich Gauss and then rediscovered by Cooley and Tukey (the radix-2 Decimation In Time or DIT case). DFT is defined by the formula:

$$\hat{f}_k = \sum_{n=0}^{N-1} f_n . e^{-\frac{2\pi i}{N} nk} \quad k = 0, 1, .., N-1$$

Radix-2 DIT first computes the DFTs of the even-indexed inputs

$$(f_{2m} = f_0, f_2, .., f_{N-2})$$

and of the odd-indexed inputs

$$(f_{2m+1} = f_1, f_3, .., f_{N-1})$$

and then combines those two results to produce the DFT of the whole sequence.

Fast Fourier Transform 1D (radix-2) - 2

$$\hat{f}_k = \sum_{m=0}^{N/2-1} f_{2m}.e^{-\frac{2\pi i}{N}(2m)k} + \sum_{m=0}^{N/2-1} f_{2m+1}.e^{-\frac{2\pi i}{N}(2m+1)k}$$

$$\hat{f}_k = \sum_{m=0}^{N/2-1} f_{2m}.e^{-\frac{2\pi i}{N/2}mk} + e^{-\frac{2\pi i}{N}k} \sum_{m=0}^{N/2-1} f_{2m+1}.e^{-\frac{2\pi i}{N/2}mk}$$

$$\hat{f}_k = E_k + e^{-\frac{2\pi i}{N}k}.O_k$$

where E_k is the DFT of the even-indexed part of f_m, and O_k is the DFT of the odd-indexed part of f_m.

THE UNIVERSITY OF CHICAGO

Fast Fourier Transform 1D (radix-2) - 3

Based on the periodicity of the DFT: $E_{k+\frac{N}{2}} = E_k$, $O_{k+\frac{N}{2}} = O_k$. We have the following:

$$\hat{f}_k = E_k + e^{-\frac{2\pi i}{N}k}.O_k \quad 0 \leq k < \frac{N}{2}$$

$$\hat{f}_k = E_{k-\frac{N}{2}} + e^{-\frac{2\pi i}{N}k}.O_{k-\frac{N}{2}} \quad \frac{N}{2} \leq k < N$$

The harmonic factor $e^{-2\pi ik/N}$ have the property that:

$$e^{-\frac{2\pi i}{N}(k+\frac{N}{2})} = e^{\frac{-2\pi ik}{N}-\pi i} = -e^{-\frac{2\pi ik}{N}}$$

We can cut the number of harmonic factor calculations in half. For $0 \leq k < \frac{N}{2}$:

$$\hat{f}_k = E_k + e^{-\frac{2\pi i}{N}k}.O_k$$

$$\hat{f}_{k+\frac{N}{2}} = E_k - e^{-\frac{2\pi i}{N}k}.O_k$$

Fast Fourier Transform 1D (radix-2) - 4

Pseudocode:

01: $\hat{f}_{0,1,..,N-1} \leftarrow FFT(f, N, s):$ *returns the DFT of* $(f_0, f_s, f_{2s}, ..)$
02: *if* $N = 1$ *then*
03: $\hat{f}_0 \leftarrow f_0$
04: *else*
05: $\hat{f}_{0,..,N/2-1} \leftarrow FFT(f, N/2, 2s)$
06: $\hat{f}_{N/2,..,N-1} \leftarrow FFT(f + s, N/2, 2s)$
07: *for* $k = 0$ *to* $N/2 - 1$
08: $t \leftarrow \hat{f}_k$
09: $\hat{f}_k \leftarrow t + exp(-2\pi ik/N)\hat{f}_{k+N/2}$
10: $\hat{f}_{k+N/2} \leftarrow t - exp(-2\pi ik/N)\hat{f}_{k+N/2}$
11: *endfor*
12: *endif*

Note that: $f + s$ (in the context of $C++$ programming language) means moving the array pointer of f to the s-th element.

THE UNIVERSITY OF CHICAGO

Fast Fourier Transform on $\mathbb{Z}_n$ - 1

DFT:

$$\hat{f}_k = \sum_{j=0}^{N-1} f_j w^{jk} \qquad k = 0, 1, .., N-1 \qquad w = e^{2\pi i/N}$$

Cooley and Tukey derived and implemented an algorithm which given a prime factorization of $N = p_1..p_r$, computed the DFT in $N\sum_i p_i$. If each $p_i = 2$, this is $2N\log_2 N$ operations. Consider $N = pq$. We change the indexing:

$$j = i_2 + i_1 q$$

$$k = m_1 + m_2 p$$

Fast Fourier Transform on $\mathbb{Z}_n$ - 2

Define the two-dimensional arrays:

$$f_{i_1,i_2} = f_j \qquad i_1 = 0,..,p-1 \qquad i_2 = 0,..,q-1$$

$$\hat{f}_{m_1,m_2} = \hat{f}_k \qquad m_1 = 0,..,p-1 \qquad m_2 = 0,..,q-1$$

DFT becomes:

$$\hat{f}_{m_1,m_2} = \sum_{i_2=0}^{q-1} w^{i_2(m_1+m_2 p)} \sum_{i_1=0}^{p-1} (w^q)^{i_1 m_1} f_{i_1,i_2}$$

Fast Fourier Transform on $\mathbb{Z}_n$ - 3

The computation is performed into two steps:

- First, q transforms of length p are computed according to

$$\overline{f}_{i_2,m_1} = \sum_{i_1=0}^{p-1} (w^q)^{i_1 m_1} f_{i_1,i_2}$$

- Next, p transforms of length q are computed according to

$$\sum_{i_2=0}^{q-1} w^{i_2(m_1+m_2 p)} \overline{f}_{i_2,m_1}$$

Instead of $(pq)^2$ operations, the above uses $(pq)(p+q)$ operations. The main idea is that we have converted a one-dimensional algorithm, in terms of indexing, into a two-dimensional algorithm.

Fast Fourier Transform on $\mathbb{Z}_n$ - 4

Let $G = \mathbb{Z}/N\mathbb{Z}$ be the group of integers modulo N. Then the characters of G are the functions $\rho_0, .., \rho_{N-1}$ defined by:

$$\rho_k(j) = w^{jk}$$

Consider:

- The sequence f_j as defining a function on $\mathbb{Z}/N\mathbb{Z}$.
- The sequence $\hat{f}_k$ as defining a function on the group of characters.

The DFT is seen as the Fourier transform on $\mathbb{Z}/N\mathbb{Z}$:

$$\hat{f}_k = \hat{f}(\rho_k) = \sum_{j \in \mathbb{Z}/N\mathbb{Z}} \rho_k(j) f_j$$

THE UNIVERSITY OF CHICAGO

Non-commutative FFTs - 1

Cooley-Tukey algorithm can be generalized to non-commutative groups based on the matrix separation of variables:

$$\hat{f}(\rho) = \sum_{y \in Y} \rho(y) \sum_{h \in H} (\rho \downarrow_H)(h) f(yh) \qquad \rho \in \mathscr{R}_G$$

where $H = \{0, q, 2q, .., (p-1)q\}$ is a subgroup of $G = \mathbb{Z}_n$ isomorphic to $\mathbb{Z}_p$, and $Y = \{0, 1, 2, .., q-1\}$. By theorem of complete reducibility:

$$(\rho \downarrow_H)(h) = T^{\dagger} \Big[\bigoplus_{\rho' \in \mathscr{R}_H(\rho)} \rho'(h) \Big] T$$

where $\mathscr{R}_H(\rho)$ is a sequence of irreducible representations of H and T is a unitary matrix.

THE UNIVERSITY OF CHICAGO

Non-commutative FFTs - 2

Given a G-module V, we say that a basis B is a **Gel'fand-Tsetlin** basis for V relative to $H < G$ if the matrix representation induced by B is **adapted** to H. When $\mathscr{R}_G$ is H-**adapted**:

$$\sum_{h \in H} (\rho \downarrow_H)(h) f(yh) = \bigoplus_{\rho' \in \mathscr{R}_H(\rho)} \sum_{h \in H} \rho'(h) f(yh)$$

Define $f_y(h) = f(yh)$:

$$\bigoplus_{\rho' \in \mathscr{R}_H(\rho)} \sum_{h \in H} \rho'(h) f_y(h) = \bigoplus_{\rho'} \hat{f}_y(\rho')$$

where $\hat{f}_y$ are now Fourier transforms over H.

Non-commutative FFTs - 3

Algorithm:

1. For each $y \in Y$, compute the Fourier transform $\{\hat{f}_y(\rho)\}_{\rho' \in \mathscr{R}_H}$ (over H) of f_y.
2. If $\rho \in \mathscr{R}_G$ decomposes into irreducibles of H in the form

$$\rho = \rho'_1 \oplus \rho'_2 \oplus .. \oplus \rho'_k$$

then assemble the matrix

$$\overline{f}_y(\rho) = \hat{f}_y(\rho'_1) \oplus \hat{f}_y(\rho'_2) \oplus .. \oplus \hat{f}_y(\rho'_k)$$

for each $y \in Y$.

3. Finally, complete the transform by computing the sums:

$$\hat{f}(\rho) = \sum_{y \in Y} \rho(y) \overline{f}_y(\rho) \qquad \rho \in \mathscr{R}_G$$

THE UNIVERSITY OF CHICAGO

Clausen's FFT for the symmetric group - 1

The first non-commutative FFT was proposed by Clausen in 1989 for Fourier transformation on the symmetric group. Clausen's algorithm follows the **matrix separation of variables** tailored to the **chain of subgroups**:

$$\mathbb{S}_1 < \mathbb{S}_2 < .. < \mathbb{S}_{n-1} < \mathbb{S}_n$$

THE UNIVERSITY OF CHICAGO

Clausen's FFT for the symmetric group - 2

Clausen's theorem. Let f be a function $\mathbb{S}_n \to \mathbb{C}$ and let

$$\hat{f}(\lambda) = \sum_{\sigma \in \mathbb{S}_n} \rho_\lambda(\sigma) f(\sigma) \qquad \lambda \vdash n$$

be its Fourier transform with respect to Young's orthogonal representation. For $i = 1, 2, .., n$, define $f_i : \mathbb{S}_{n-1} \to \mathbb{C}$ as $f_i(\sigma') = f([\![i, n]\!])\sigma')$ for $\sigma' \in \mathbb{S}_{n-1}$, and let

$$\hat{f}_i(\lambda^-) = \sum_{\sigma' \in \mathbb{S}_{n-1}} \rho_{\lambda^-}(\sigma') f_i(\sigma') \qquad \lambda^- \vdash n-1$$

be the corresponding Fourier transforms, again with respect to Young's orthogonal representation. Then up to reordering of rows and columns:

$$\hat{f}(\lambda) = \sum_{i=1}^{n} \rho_\lambda([\![i, n]\!]) \bigoplus_{\lambda^- \in R(\lambda)} \hat{f}_i(\lambda^-)$$

where $R(\lambda) = \{\lambda^- \vdash n-1 | \lambda^- \leq \lambda\}$.

THE UNIVERSITY OF CHICAGO

Fast Fourier Transform 2D - 1

Discrete Fourier Transform 2D:

$$\hat{f}_{x,y} = \frac{1}{\sqrt{MN}} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} f_{u,v}.e^{-i2\pi\left(\frac{xu}{M}+\frac{yv}{N}\right)}$$

$$\hat{f}_{x,y} = \frac{1}{\sqrt{MN}} \sum_{u=0}^{M-1} e^{-i2\pi\frac{xu}{M}}.\left(\sum_{v=0}^{N-1} f_{u,v}.e^{-i2\pi\frac{yv}{N}}\right)$$

We can compute $P(u,y) = \sum_{v=0}^{N-1} f_{u,v}.e^{-i2\pi\frac{yv}{N}}$ by FFT-1D. Again, we use FFT-1D to compute $\hat{f}_{x,y} = \frac{1}{\sqrt{MN}} \sum_{u=0}^{M-1} e^{-i2\pi\frac{xu}{M}}.P(u,y)$.

Fast Fourier Transform 2D - 2

Inverse Discrete Fourier Transform 2D:

$$f_{u,v} = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \hat{f}_{x,y}.e^{i2\pi\left(\frac{xu}{M}+\frac{yv}{N}\right)}$$

$$f_{u,v} = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} e^{i2\pi\frac{xu}{M}}.\left(\sum_{y=0}^{N-1} \hat{f}_{x,y}.e^{i2\pi\frac{yv}{N}}\right)$$

Compute $H(x,v) = \sum_{y=0}^{N-1} \hat{f}_{x,y}.e^{i2\pi\frac{yv}{N}}$ and then $f_{u,v} = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} e^{i2\pi\frac{xu}{M}}.H(x,v)$ by FFT-1D algorithm.

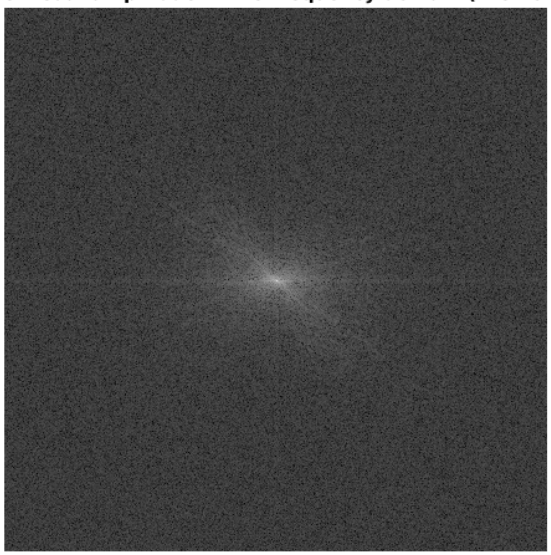
THE UNIVERSITY OF CHICAGO

The effect of Gaussian filter on noisy image - 1

example_fft_2d_gaussian_denoising_by_smoothing.m

The noised image with the gaussian noise (magnitude = 20)

The visual amplitude in the frequency domain (the noised image)

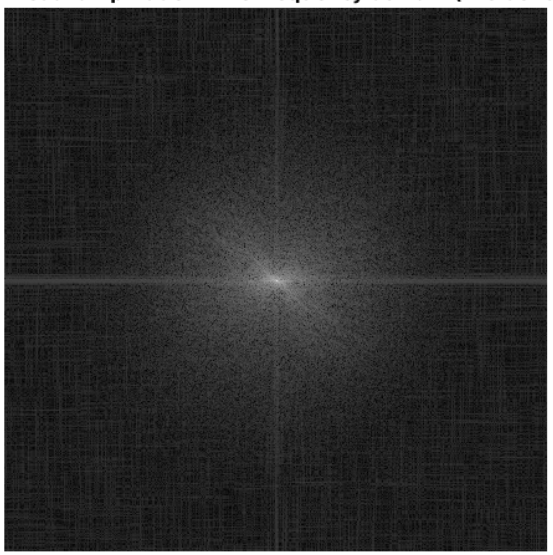


The effect of Gaussian filter on noisy image - 2

example_fft_2d_gaussian_denoising_by_smoothing.m

The denoised image by Gaussian matrix (radius = 5, variance = 1)

The visual amplitude in the frequency domain (the denoised image)

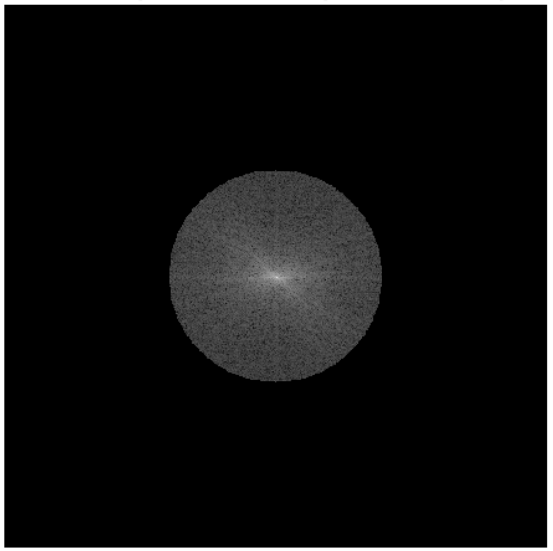


Low-pass filter

example_fft_2d_gaussian_denoising_by_lowpass_filter.m
example_fft_2d_gaussian_smoothing.m

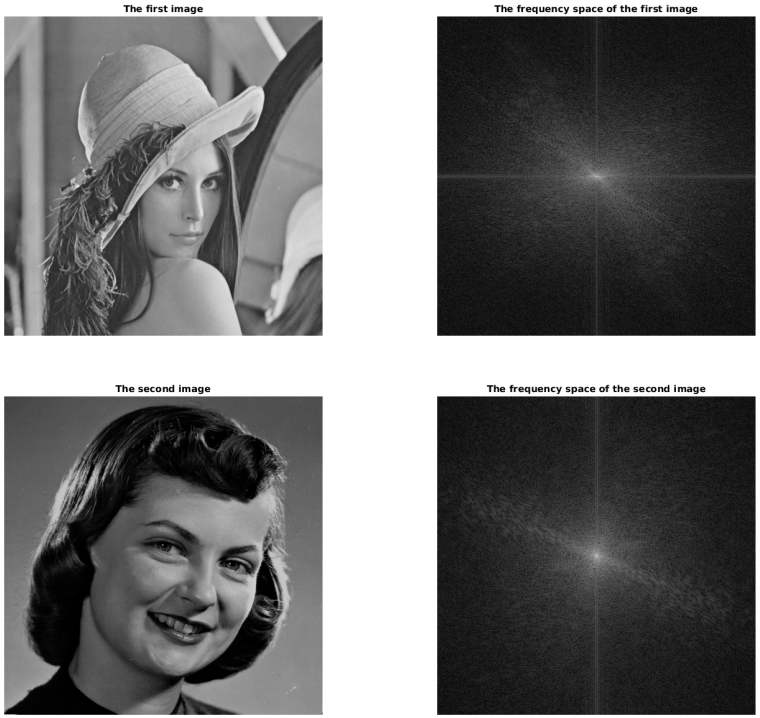
The lowpass-filtered image

The visual amplitude in the lowpass-filtered frequency domain



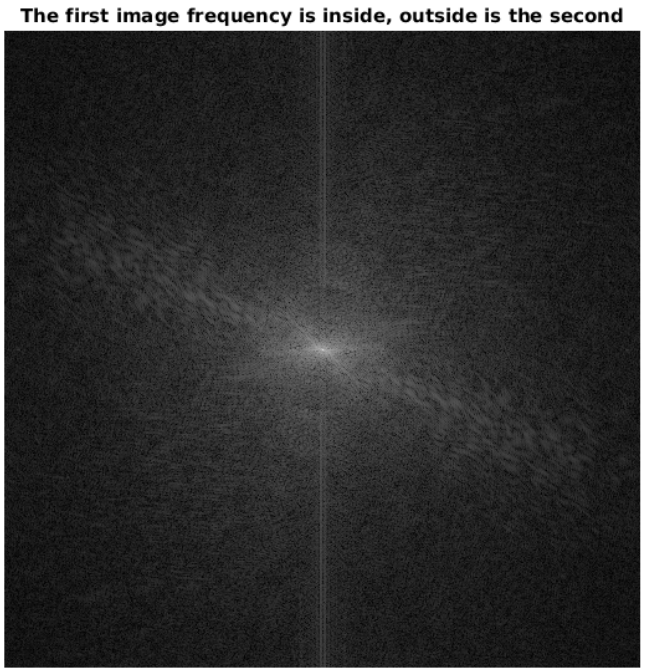
Hybrid image - Mixing frequencies - 1

example_fft_2d_hybrid_frequecy_image.m



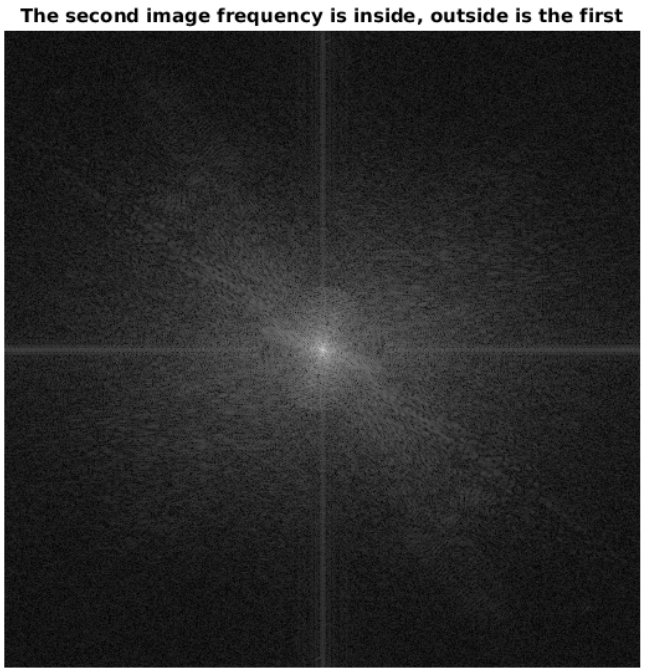
Hybrid image - Mixing frequencies - 2

example_fft_2d_hybrid_frequecy_image.m



Hybrid image - Mixing frequencies - 3

example_fft_2d_hybrid_frequecy_image.m



Discrete Cosine Transform

Discrete Cosine Transform:

$$\hat{f}_{u,v} = \frac{1}{4}\alpha(u)\alpha(v)\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f_{x,y} cos\Big[\frac{(2x+1)u\pi}{2M}\Big] cos\Big[\frac{(2y+1)v\pi}{2N}\Big]$$

where $\alpha(u) = \frac{1}{\sqrt{2}}$ if $u = 0$, and $\alpha(u) = 1$ otherwise.
Inverse Discrete Cosine Transform:

$$f_{x,y} = \frac{1}{4}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} \hat{f}_{u,v}.\alpha(u)\alpha(v) cos\Big[\frac{(2x+1)u\pi}{2M}\Big] cos\Big[\frac{(2y+1)v\pi}{2N}\Big]$$

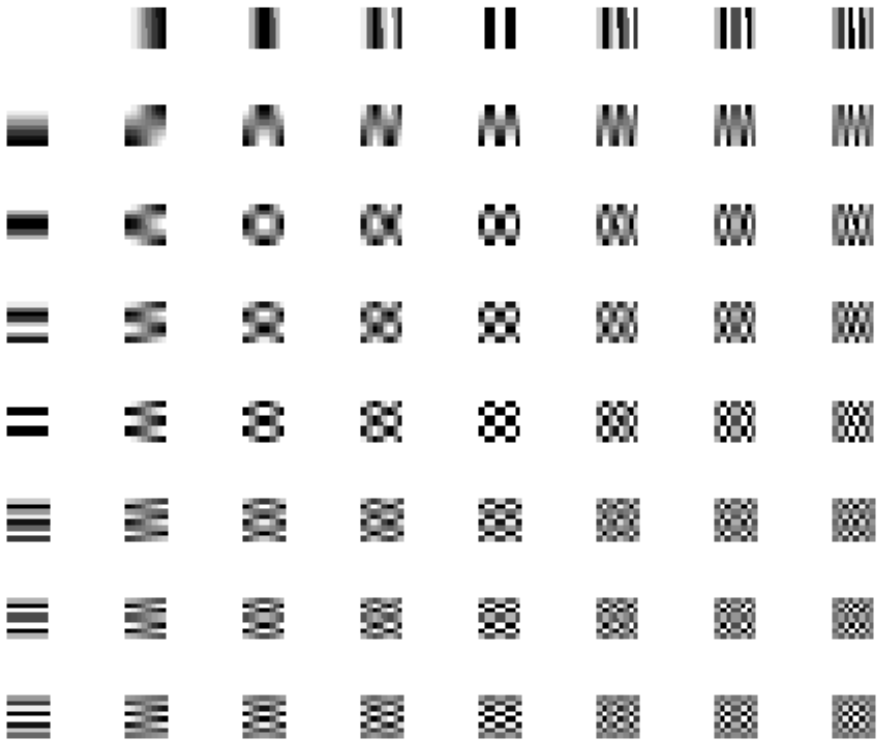
Source code:

dct_2d_jpeg.cpp

THE UNIVERSITY OF CHICAGO

DCT 2D operators

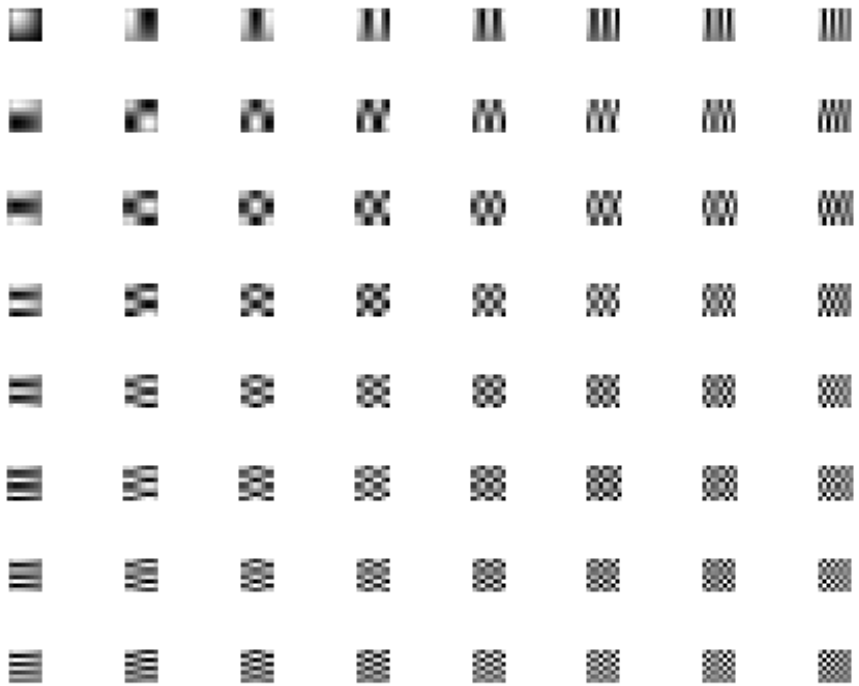
example_dct_2d_operator.m



THE UNIVERSITY OF CHICAGO

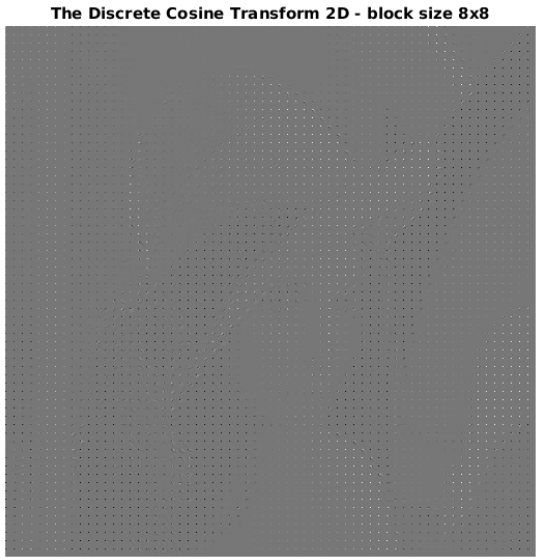
Inverse DCT 2D operators

example_dct_2d_operator.m



JPEG by DCT 2D with Huffman code

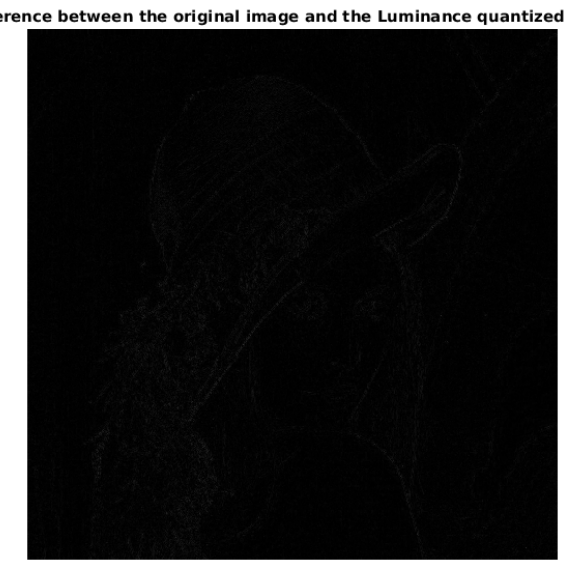
example_dct_2d_jpeg.m



THE UNIVERSITY OF CHICAGO

Luminance quantization for JPEG

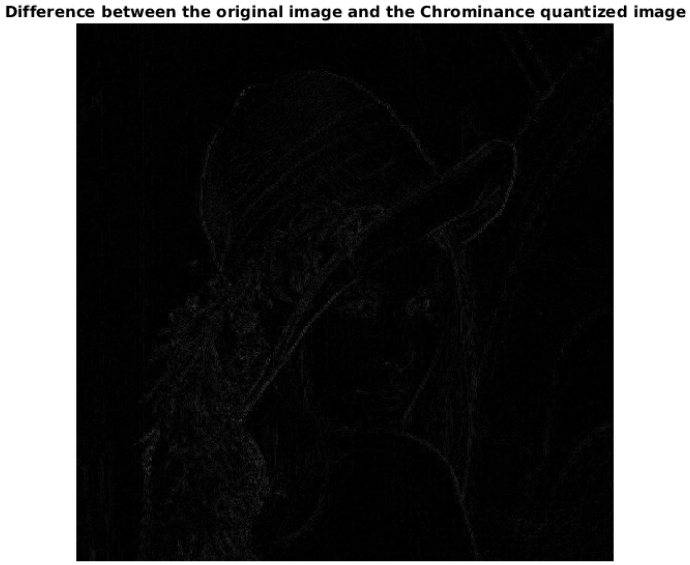
example_dct_2d_jpeg.m



THE UNIVERSITY OF CHICAGO

Chrominance quantization for JPEG

example_dct_2d_jpeg.m



THE UNIVERSITY OF CHICAGO

Fast Polynomial Multiplication by FFT - 1

Given two polynomials:

$$A(x) = \sum_{i=0}^{n-1} a_i.x^i$$

$$B(x) = \sum_{i=0}^{m-1} b_i.x^i$$

Compute:

$$C(x) = A(x).B(x) = \sum_{i=0}^{k-1} c_i.x^i \quad (k = n+m-1)$$

where $c_i = \sum_j a_j.b_{i-j}$.

THE UNIVERSITY OF CHICAGO

Fast Polynomial Multiplication by FFT - 2

Let N be the smallest power of 2 that is bigger than or equal to k. We can rewrite $A(x)$, $B(x)$, $C(x)$ as polynomials of degree $N-1$ as follows:

$$A(x) = \sum_{i=0}^{n-1} a_i.x^i + \sum_{i=n}^{N-1} 0.x^i$$

$$B(x) = \sum_{i=0}^{m-1} b_i.x^i + \sum_{i=m}^{N-1} 0.x^i$$

$$C(x) = \sum_{i=0}^{k-1} c_i.x^i + \sum_{i=k}^{N-1} 0.x^i$$

Fast Polynomial Multiplication by FFT - 3

Let w be the N-th complex root of unity: $w = cos(\frac{2\pi}{N}) + i.sin(\frac{2\pi}{N})$. Some properties of w:

$$w^j = \left[cos\left(\frac{2\pi}{N}\right) + i.sin\left(\frac{2\pi}{N}\right)\right]^j = cos\left(\frac{2\pi}{N}j\right) + i.sin\left(\frac{2\pi}{N}j\right)$$

$$w^0 = 1$$

$$w^N = 1$$

$$w^{j+\frac{N}{2}} = -w^j$$

$$w^{j+N} = w^j$$

THE UNIVERSITY OF CHICAGO

Fast Polynomial Multiplication by FFT - 4

We will use the idea of the Fast Fourier Transform algorithm to compute the values of $A(x)$ and $B(x)$ at this special series $W = \{w^0, w^1, w^2, .., w^{N-1}\}$ in time complexity $O(N \log N)$. Then we can compute the values of $C(x)$ at W in time complexity $O(N)$. Finally, from these values, we can intepolate to compute the coefficients of $C(x)$ again by FFT.

Let $P(x)$ be an arbitrary polynomial of degree $N-1$: $P(x) = \sum_{i=0}^{N-1} p_i.x^i$. We have:

$$P(w^j) = \sum_{i=0}^{N-1} p_i.(w^j)^i$$

$$\Leftrightarrow P(w^j) = \sum_{k=0}^{N/2-1} p_{2k}.(w^j)^{2k} + \sum_{k=0}^{N/2-1} p_{2k+1}.(w^j)^{2k+1}$$

$$\Leftrightarrow P(w^j) = \sum_{k=0}^{N/2-1} p_{2k}.(w^{2j})^{k} + w^j.\sum_{k=0}^{N/2-1} p_{2k+1}.(w^{2j})^{k}$$

Fast Polynomial Multiplication by FFT - 5

On the another hand:

$$P(w^{j+N/2}) = \sum_{k=0}^{N/2-1} p_{2k}.(w^{2j+N})^k + w^{j+N/2}.\sum_{k=0}^{N/2-1} p_{2k+1}.(w^{2j+N})^k$$

$$\Leftrightarrow P(w^{j+N/2}) = \sum_{k=0}^{N/2-1} p_{2k}.(w^{2j})^k - w^j.\sum_{k=0}^{N/2-1} p_{2k+1}.(w^{2j})^k$$

Fast Polynomial Multiplication by FFT - 6

Pseudocode of computing the DFT $F(P)$ or the evaluation of $P(x)$ at $1, w, w^2, .., w^{N-1}$:

01: $F_{0,1,..,N-1} \leftarrow FFT(p, N, s, w)$: *returns the DFT of* $(p_0, p_s, p_{2s}, ..)$
02: *if* $N = 1$ *then*
03: $F_0 \leftarrow p_0$
04: *else*
05: $F_{0,..,N/2-1} \leftarrow FFT(p, N/2, 2s, w^2)$
06: $F_{N/2,..,N-1} \leftarrow FFT(p+s, N/2, 2s, w^2)$
07: $x \leftarrow 1$
08: *for* $j = 0$ *to* $N/2 - 1$
09: $t \leftarrow F_j$
10: $F_j \leftarrow t + x.F_{j+N/2}$
11: $F_{j+N/2} \leftarrow t - x.F_{j+N/2}$
12: $x \leftarrow x.w$
13: *endfor*
14: *endif*

Fast Polynomial Multiplication by FFT - 7

The final algorithm is:

- Step 1. Fourier transform for evaluation - time $O(N \log N)$:
  $F_A \leftarrow FFT(A, N, 1, w)$
- Step 2. Fourier transform for evaluation - time $O(N \log N)$:
  $F_B \leftarrow FFT(B, N, 1, w)$
- Step 3. Time $O(N)$:
  for $i = 1$ *to* N
  $F_C[i] \leftarrow F_A[i].F_B[i]$
  endfor
- Step 4. Fourier Transform for interpolation - time $O(N \log N)$:
  $C \leftarrow \frac{1}{N}.FFT(F_C, N, 1, w^{-1})$

Note that $w^{-1} = cos(\frac{2\pi}{N}) - i.sin(\frac{2\pi}{N})$

THE UNIVERSITY OF CHICAGO

Fast Integer Multiplication by FFT

Given two decimal integer $A = (a_{n-1}, .., a_1, a_0)$ and $B = (b_{m-1}, .., b_1, b_0)$. A and B can be expressed in terms of polynomials:

$$A(10) = \sum_{i=0}^{n-1} a_i.10^i$$

$$B(10) = \sum_{i=0}^{m-1} b_i.10^i$$

Then the product of A and B is equal to $A(10).B(10)$. Apply the Fast Fourier Transform algorithm as described above, we can compute the multiplication polynomial $C(x) = A(x).B(x)$ faster, so the result will be $C(10)$.

Source code:

polymul_fft.cpp, example_polymul_fft.m, intmul_fft.cpp, example_intmul_fft.m

THE UNIVERSITY OF CHICAGO

Q & A

Thank you very much for your attention!

THE UNIVERSITY OF CHICAGO