Image Filtering and the Fourier Transform

Steven Tanimoto

CSE 457
Spring 2012

Reading

Recommended:

- Tanimoto. "Filtering", in *An Interdisciplinary Introduction to Image Processing: Pixels, Numbers, and Programs,* Sections 10.10-10.14.

Motivation in terms of Convolution

Image filtering: convolution is the most common approach.

Computational cost: convolution of an n by n image with an m by m kernel ?

Direct Method requires n^2m^2 multiplications and additions.

If m is on the order of n, then this is $\Omega(n^4)$.

With a "smart" approach, we can get this down to $O(n^2 \log n)$.

The smart approach makes use of the Discrete Fourier Transform.

Other motivation: the Fourier Transform is a fundamental mathematical concept in functional analysis and signal/image processing.

Uses for Fourier Transforms

Analyze sound: speech, music, sounds from nature, sounds from machines

Synthesize sound, filter sound

Filter images for image enhancement

Filter image for image analysis

Analyze the optical properties of cameras and scanners

Measure shape of objects and characteristics of textures in images

Bases for Vector Spaces

By using different bases, the same information (vector) can be expressed in different ways.

This could be used to encrypt an image, for example.

But it can be particularly helpful for tasks like filtering out various frequency components and for image analysis.

Before we can explain the Fourier Transform, we need to consider bases that involve not just real numbers, but complex numbers.

Complex Numbers

$c = a + b\,i$

c is a complex number
a and b are real numbers

$i = \sqrt{-1}$

Originally invented in order to solve polynomial equations like

$$x^2 + 1 = 0$$

Complex Exponentials

Something particularly useful in Fourier transforms.

e = 2.71828… $e = \sum_{n=0}^{\infty} \frac{1}{n!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots$

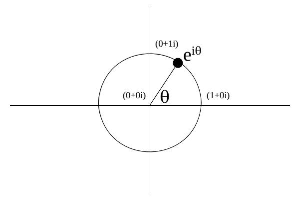
e^x is an exponentially growing function, where x is real.

$e^{i\theta}$ is an oscillating function. Here θ is the phase angle.

$e^{ni\theta}$ for n > 1 are also oscillating functions. Their frequencies of oscillation are harmonics of that of $e^{i\theta}$

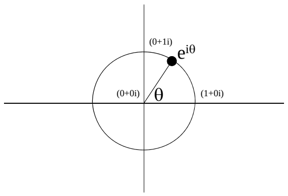
Complex Exponentials (cont.)

Graph of $e^{i\theta}$
It's the "unit circle"



Complex Exponentials (cont.)

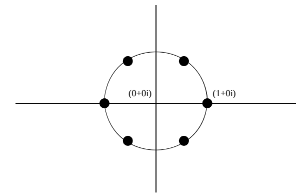
Graph of $e^{i\theta}$
It's the "unit circle"



Leonard Euler's amazing identity: $e^{i\pi} + 1 = 0$

Complex *n*th roots of 1

The *N*th roots of 1 are $e^{2ki\pi/N}$ for k = 0, 1, 2, …, N-1.
ω is a **principal *N*th root of unity** if it is an *N*th root of 1 (unity) and all the other *N*th roots of 1 are powers of ω.



Example with N = 6 $\theta = 2k\,\pi / N$

Separating the Real and Imaginary Parts of $e^{i\theta}$

$$\text{real}(e^{i\theta}) = \cos\theta$$

$$\text{imag}(e^{i\theta}) = \sin\theta$$

Definition of the Discrete Fourier Transform

Let S be a sequence of N complex numbers:
$S = [c_0, c_1, \ldots, c_{N-1}] =$
$[a_0+b_0 i, a_1+b_1 i, \ldots, a_{N-1}+b_{N-1} i]$

The **Discrete Fourier Transform** DFT(S) is defined as follows

DFT(S) $= [f_0, f_1, \ldots, f_{N-1}]$, where

$$f_n = \sum_{k=0,N-1} c_k\, e^{2ki\pi n / N}$$

Thus each element f_n of DFT(S) is a weighted sum of elements of S,
but each weight is a complex *N*th root of 1.

Definition of the Discrete Fourier Transform

Let's enlarge the key formula.

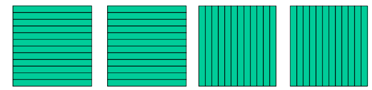
$$f_n = \sum_{k=0}^{N-1} c_k \, e^{2ki\pi n / N}$$

13

Fourier Transforms in Image Processing

Since an image is 2-dimensional, we usually apply the DFT to an image in two passes: **row transforms**, and then **column transforms**.
(The order of these should not matter, according to theory.)

Row transforms — Column transforms



14

Filtering with the DFT

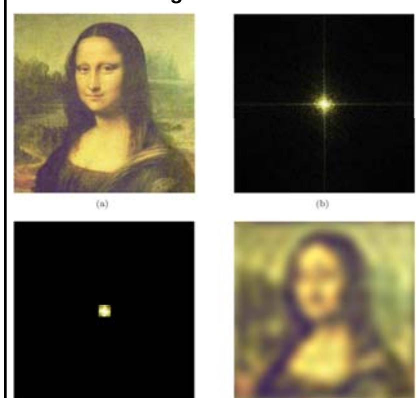
Make sure image is 2^k by 2^k in size, and represented as complex.
Perform the 2D DFT
(Optional: adjust coordinate system of transform image)
"Edit" the transform image
(Optional: restore coordinate system of transform image)
Perform the inverse 2D DFT
(use opposite sign for each weighting factor's exponent,
and divide result by 1/N in each pass.)

In some cases, *editing* means setting some frequency components to zero.
In other cases, it means multiplying the FT of the image by the FT of the convolution kernel.

The Inverse 2D DFT is equivalent to the (forward) 2D DFT, except that at the end, each element must be divided by a scale factor N^2. This can be done as division by N after the inverse row transforms and another division by N after the inverse column transforms, or all at the end.

15

Filtering with the DFT



Original, 2DDFT,
edited 2DDFT, inverse-2DDFT result.

16

Note: There is a Fast Fourier Transform

The “Fast Fourier Transform” (FFT) performs the summation of weighted pixel values in a clever manner, saving a lot of time. It’s based on the fact that the weights (Nth roots of unity) are all related.

Note: Key Ideas behind the FFT

Computing the DFT at a particular value x is equivalent to **evaluating a polynomial at x**.

Given the input:

$$[f_0, f_1, \ldots, f_{N-1}]$$

we just need to evaluate

$$P(x) = f_0 x^0 + f_1 x^1 + \cdots + f_{N-1} x^{N-1}$$

The important values of x are the powers of ω which is a principal root of unity. $\omega = e^{-2\pi i/N}$

$$1, \omega, \omega^2, \ldots, \omega^{N-1}$$

The DFT of the input will be equivalent to

[$P(1)$, $P(\omega)$, $P(\omega^2)$, . . . , $P(\omega^{N-1})$]

Note: Key Ideas: Divide and Conquer

The polynomial P(x) can be split into two parts:

$$P(x) = f_0 x^0 + f_1 x^1 + \cdots + f_{N-1} x^{N-1}$$

$$\begin{aligned} P_{\text{even}}(x) &= f_0 x^0 + f_2 x^2 + \cdots + f_{N-2} x^{N-2} \\ P_{\text{odd}}(x) &= f_1 x^1 + f_3 x^3 + \cdots + f_{N-1} x^{N-1} \end{aligned}$$

$$P(x) = P_{\text{even}}(x) + P_{\text{odd}}(x)$$

With some minor modifications we turn each of P_{even} and P_{odd} which are "sparse" polynomials into Q_{even} and Q_{odd}, which are normal polynomials but of degree N/2.

$$Q_{\text{even}}(y) = f_0 y^0 + f_2 y^1 + \cdots + f_{N-2} y^{N/2-1}$$

$$Q_{\text{odd}}(y) = f_1 y^0 + f_3 y^1 + \cdots + f_{N-1} y^{N/2-2}$$

This gives us a recurrence that we can use to good effect:

$$P(x) = Q_{\text{even}}(x^2) + x Q_{\text{odd}}(x^2)$$

Note: Key Ideas: 2 for the price of 1

Whenever we compute $P(\omega^k)$ we also compute $P(\omega^{k+N/2})$

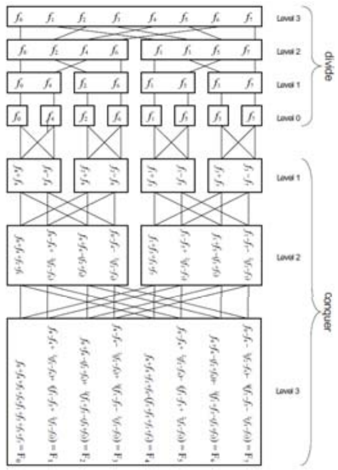
$$P(\omega^k) = Q_{\text{even}}(\omega^{2k}) + \omega^k Q_{\text{odd}}(\omega^{2k})$$

$$P(\omega^{k+N/2}) = Q_{\text{even}}(\omega^{2k}) - \omega^k Q_{\text{odd}}(\omega^{2k})$$

One instance of computing these two values is called a "**butterfly step**".

If we unwind the entire recursive computation, we can see a data flow graph of the whole operation.

Data Flow Diagram for the FFT



Computational Complexity of the FFT

Each butterfly step requires one multiplication and two additions.

There are $n/2 \log_2 n$ butterfly steps.

Thus the FFT requires $\Theta(n \log n)$ time.

Cost of Convolution of an n by n image with another n by n image.

This is a somewhat hypothetical example, because, without padding, we are assuming that the images are sections of periodic 2D functions.

(More typical: the kernel is smaller, and some padding is added to the big image.)

1. Perform the 2DDFT of the rows of each image.
2. Perform the 2DDFT of the columns of the row-transformed images.
3. Multiply the two resulting images, pixel-by-pixel.
4. Perform inverse column transformations on the product image.
5. Perform inverse row transformations on the previous result

Time required:
$O(2n(n \log n))$ time for each of steps 1, 2, 3, 4.
$O(n^2)$ for step 5.
Overall: $O(n^2 \log n)$

Summary

What to take home:

- The meanings of all the **boldfaced** terms.
- How to perform convolution using the Fourier transform.
- Vector basis used by the Discrete Fourier transform.
- Key ideas in the Fast Fourier Transform.