# Fast 3D Convolution algorithms summary

A. Kouzelis, Prof. Ioannis Pitas
Aristotle University of Thessaloniki
pitas@csd.auth.gr

Version 3.0.2

Date 30/01/2021

# Fast 3D Convolution algorithms

- 3D linear and cyclic convolutions
- Fast 3D convolutions by using FFTs
- Block-based methods
- Optimal Winograd 3D convolutions

# Introduction

- Convolution plays a very important role in image/video processing and analysis, machine learning etc.
- Convolutional neural networks (CNNs) are based on the convolution (they form the first layers).
- Computationally expensive, $O(N^6)$ in 3D.
- There is a need for efficient convolution algorithms.

# 3D Signals and Systems

- A 3D ***signal*** is a mapping of the form:

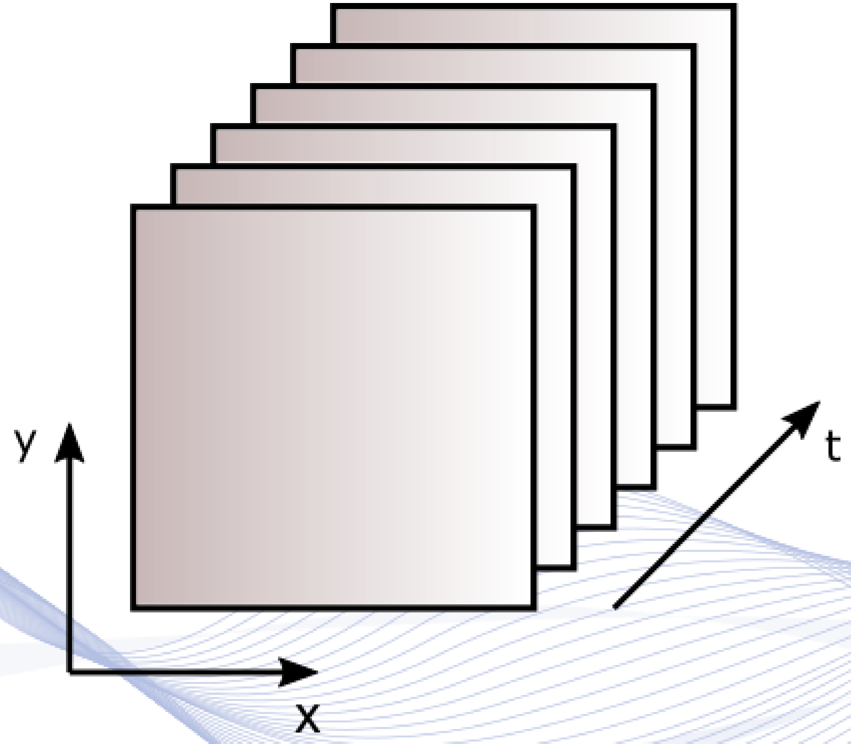
$$f_a : \mathbb{R}^3 \rightarrow \mathbb{R}$$

- The discrete version is:

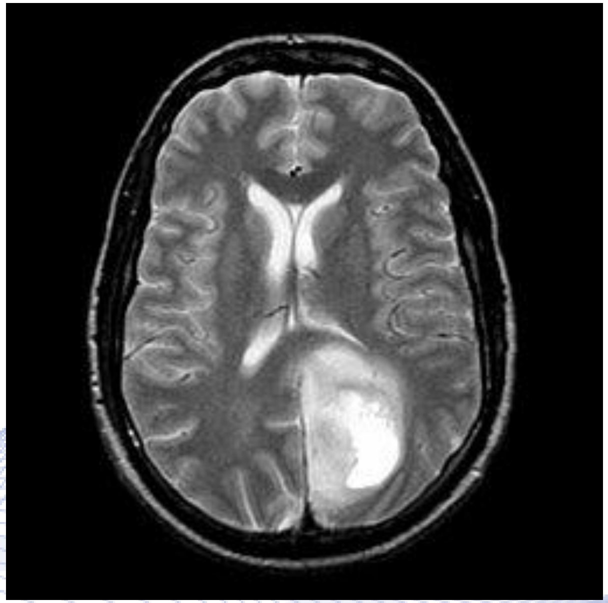
$$f : \mathbb{Z}^3 \rightarrow \mathbb{R}$$

- For example:
  - Digital video signal: $f(n_1, n_2, n_3) = f_a(n_1 \Delta x, n_2 \Delta y, n_3 \Delta t)$
  - 3D volumetric image: $f(n_1, n_2, n_3) = f_a(n_1 \Delta x, n_2 \Delta y, n_3 \Delta z)$
  - $\Delta x, \Delta y, \Delta z$ are spatial sampling intervals and $\Delta t$ is the temporal sampling interval.

# 3D Signals and Systems




Spatiotemporal video signal



2D slice of a 3D MRI image [WIK-MRI]

# 3D Linear Convolution

For a 3D LSI system with impulse response $h$, the input $x$ and output $y$ are related by the ***3D linear convolution***:

$$y(n_1, n_2, n_3) = x(n_1, n_2, n_3) *** h(n_1, n_2, n_3)$$

$$= \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} \sum_{k_3=-\infty}^{\infty} x(k_1, k_2, k_3) h(n_1 - k_1, n_2 - k_2, n_3 - k_3)$$

It is commutative: $x *** h = h *** x$.

Artificial Intelligence & Information Analysis Lab

# 3D Linear Convolution

If the system's impulse response $h$ is of finite size $N_{h_1} \times N_{h_2} \times N_{h_3}$, the system is called ***Finite Impulse Response*** (FIR) system and is described by:

$$y(n_1, n_2, n_3) = x(n_1, n_2, n_3) *** h(n_1, n_2, n_3)$$

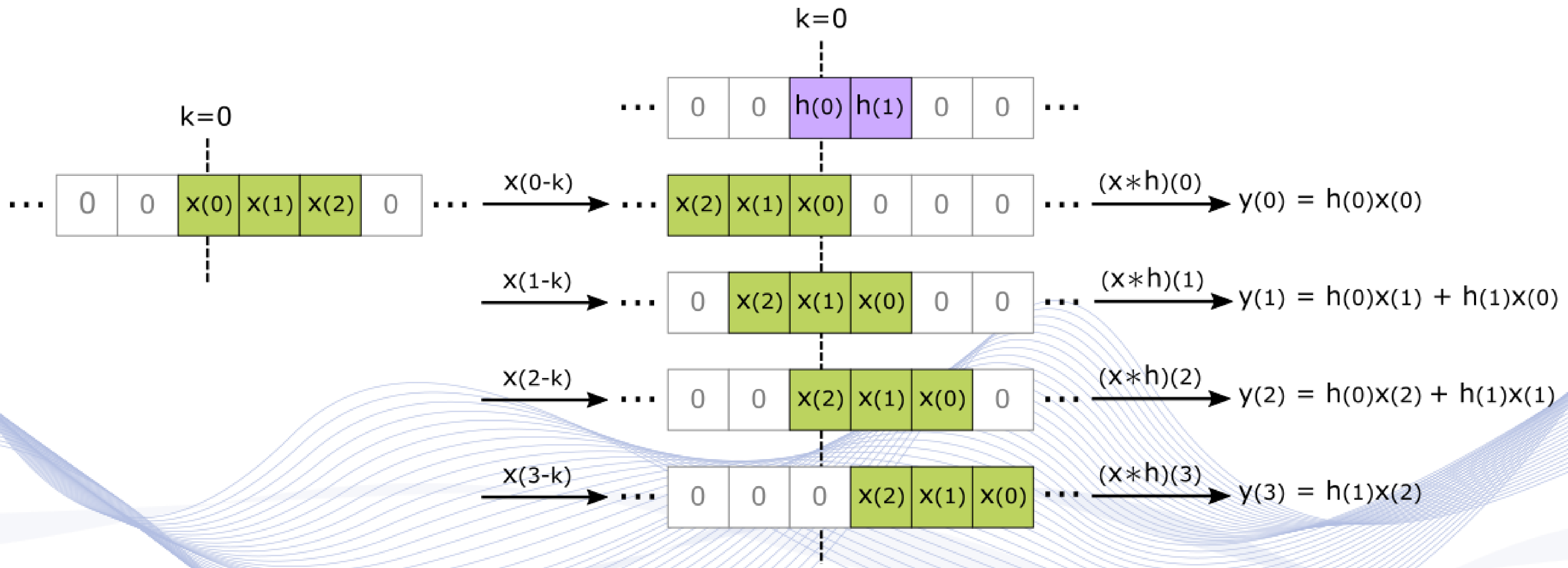
$$= \sum_{k_1=0}^{N_{h_1}-1} \sum_{k_2=0}^{N_{h_2}-1} \sum_{k_3=0}^{N_{h_3}-1} h(k_1, k_2, k_3) x(n_1 - k_1, n_2 - k_2, n_3 - k_3)$$

# 3D Linear Convolution

If the input signal $x(n_1, n_2, n_3)$ is also finite, $N_{x_1} \times N_{x_2} \times N_{x_3}$, the resulting output signal $y = x *** h$ has size:

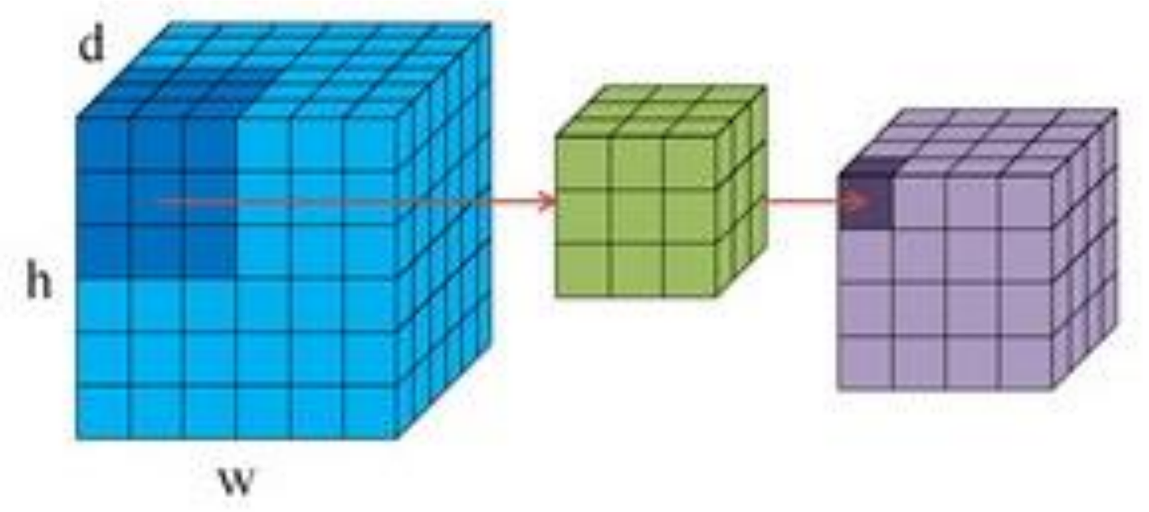
$$(N_{x_1} + N_{h_1} - 1) \times (N_{x_2} + N_{h_2} - 1) \times (N_{x_3} + N_{h_3} - 1).$$

# 3D Linear Convolution



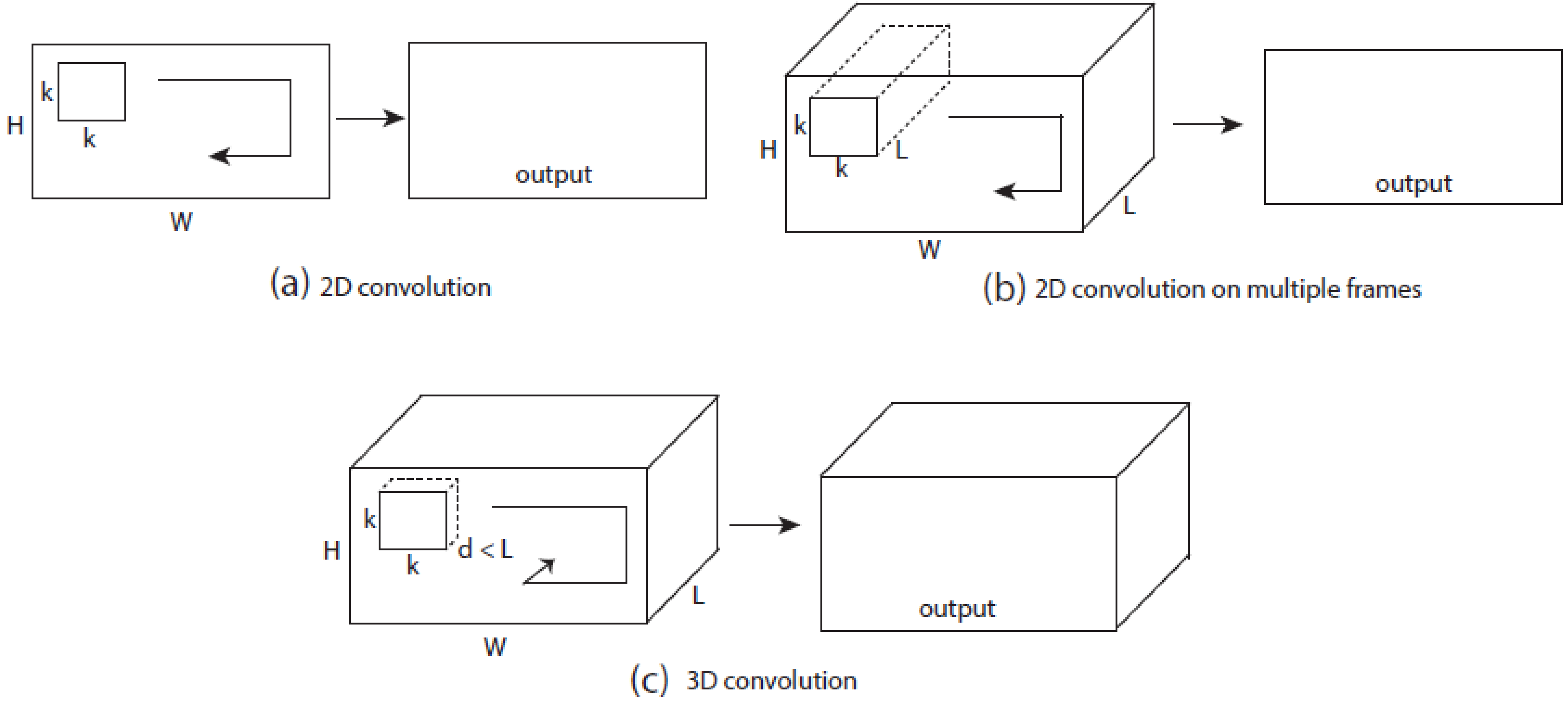

Example of 1D linear convolution: The signals $x(n)$ and $h(n)$ of finite size $N_x = 3$ and $N_h = 2$ respectively, and the output signal $y(n)$ is of size $N_x + N_h - 1 = 4$.

# 3D Linear Convolution




An illustration of 3D convolution with a kernel of size $3 \times 3 \times 3$ (from [DON2020]).

# 3D Linear Convolution




2D convolution vs 3D convolution (from [TRA2015]).

# 3D Cyclic Convolution

The ***3D cyclic convolution*** is defined as:

$$y(n_1,n_2,n_3) = x(n_1,n_2,n_3) \circledast\circledast\circledast h(n_1,n_2,n_3)$$

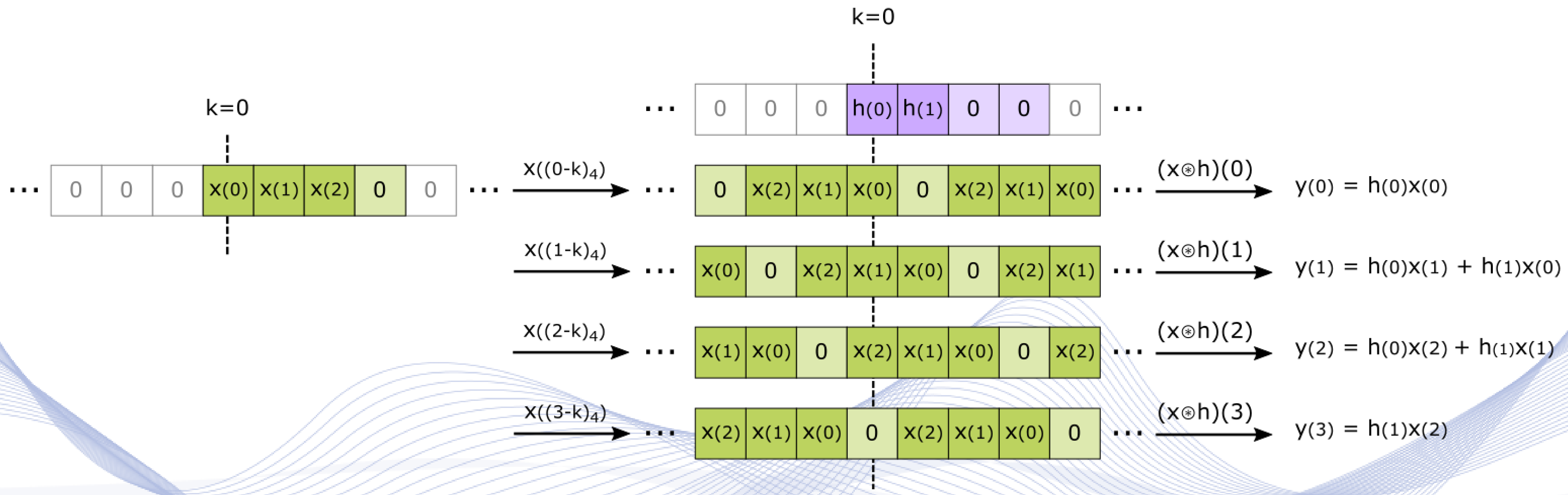
$$= \sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1}\sum_{k_3=0}^{N_3-1} x(k_1,k_2,k_3)h\left((n_1-k_1)_{N_1},(n_2-k_2)_{N_2},(n_3-k_3)_{N_3}\right)$$

where $(n)_N$ denotes $n \bmod N$ and is the ***cyclic shift***. We use the symbol $\circledast$ to distinguish it from the linear convolution.

# 3D Cyclic Convolution

- 3D linear convolution can be embedded in 3D cyclic convolution by zero-padding the $x(n_1, n_2, n_3)$ and $h(n_1, n_2, n_3)$ in each dimension (see picture on next slide).
- Performing cyclic convolution on these padded signals is equivalent to performing linear convolution on the original signals.
- Cyclic convolutions are useful because they can be computed using DFT (via FFT algorithms) and other fast algorithms such as Winograd convolution algorithms.

# 3D Cyclic Convolution




Example of 1D cyclic convolution which is equivalent to linear convolution. The original signals $x(n)$ and $h(n)$ are of size $N_x = 3$ and $N_h = 2$ respectively. By zero-padding them to the same size $N_x + N_h - 1 = 4$, the resulting output signal $y(n)$ (of size $4$) is the same as $y(n)$ obtained from linear convolution of the original signals.

# 3D Z-transform

- The ***3D Z-transform*** of a $N_1 \times N_2 \times N_3$ signal $x$ is defined as:

$$X(z_1, z_2, z_3) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \sum_{n_3=0}^{N_3-1} x(n_1, n_2, n_3) z_1^{-n_1} z_2^{-n_2} z_3^{-n_3}$$

  where $z_1, z_2, z_3$ are complex variables.
- It can be considered as a polynomial of three variables $z_1, z_2, z_3$, by multiplying it by the monomial $z_1^{N_1-1} z_2^{N_2-1} z_3^{N_3-1}$.

# 3D Z-transform

- An important property is that the 3D linear convolution is equivalent to the polynomial products in the z-domain:

$$y(n_1,n_2,n_3) = x(n_1,n_2,n_3) *** h(n_1,n_2,n_3)$$
$$\leftrightarrow Y(z_1,z_2,z_3) = X(z_1,z_2,z_3)H(z_1,z_2,z_3)$$

- Similarly for the 3D cyclic convolution:

$$y(n_1,n_2,n_3) = x(n_1,n_2,n_3) \circledast\circledast\circledast\, h(n_1,n_2,n_3)$$
$$\leftrightarrow Y(z_1,z_2,z_3) = X(z_1,z_2,z_3)H(z_1,z_2,z_3) \bmod (z_1^{N_1}-1), (z_2^{N_2}-1), (z_3^{N_3}-1)$$

# 3D Discrete Fourier Transform

The ***3D Discrete Fourier Transform (DFT)*** of a 3D $N_1 \times N_2 \times N_3$ signal $x$ is defined as:

$$X(k_1, k_2, k_3) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \sum_{n_3=0}^{N_3-1} x(n_1, n_2, n_3) W_{N_1}^{n_1 k_1} W_{N_2}^{n_2 k_2} W_{N_3}^{n_3 k_3}$$

where $W_{N_i} \equiv e^{-j2\pi/N_i}$, $i = 1,2,3$, are $N_i$-th ***primitive roots of unity***.

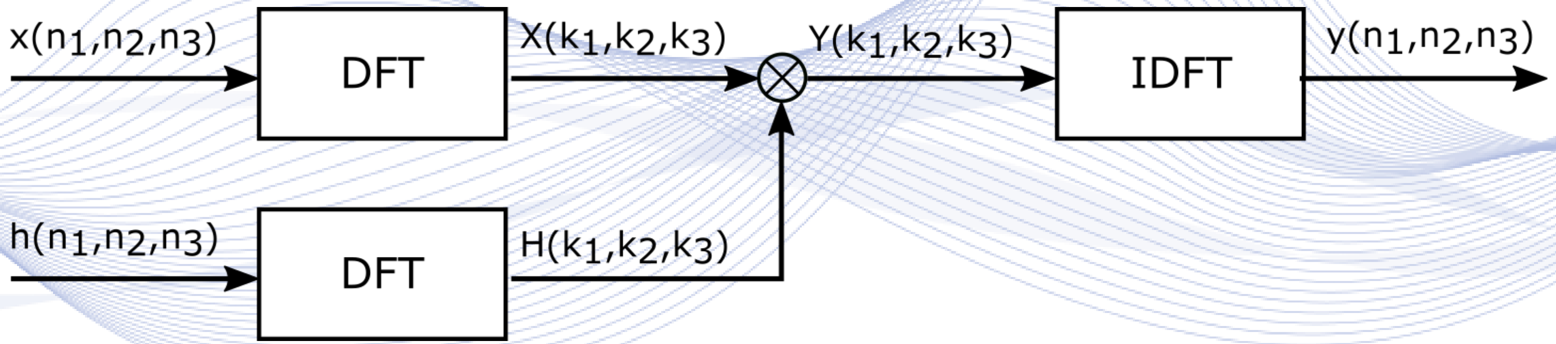
# 3D Discrete Fourier Transform

- The ***convolution theorem*** states that the cyclic convolution in $\mathbb{Z}^3$ is equivalent to the multiplication in the DFT domain:

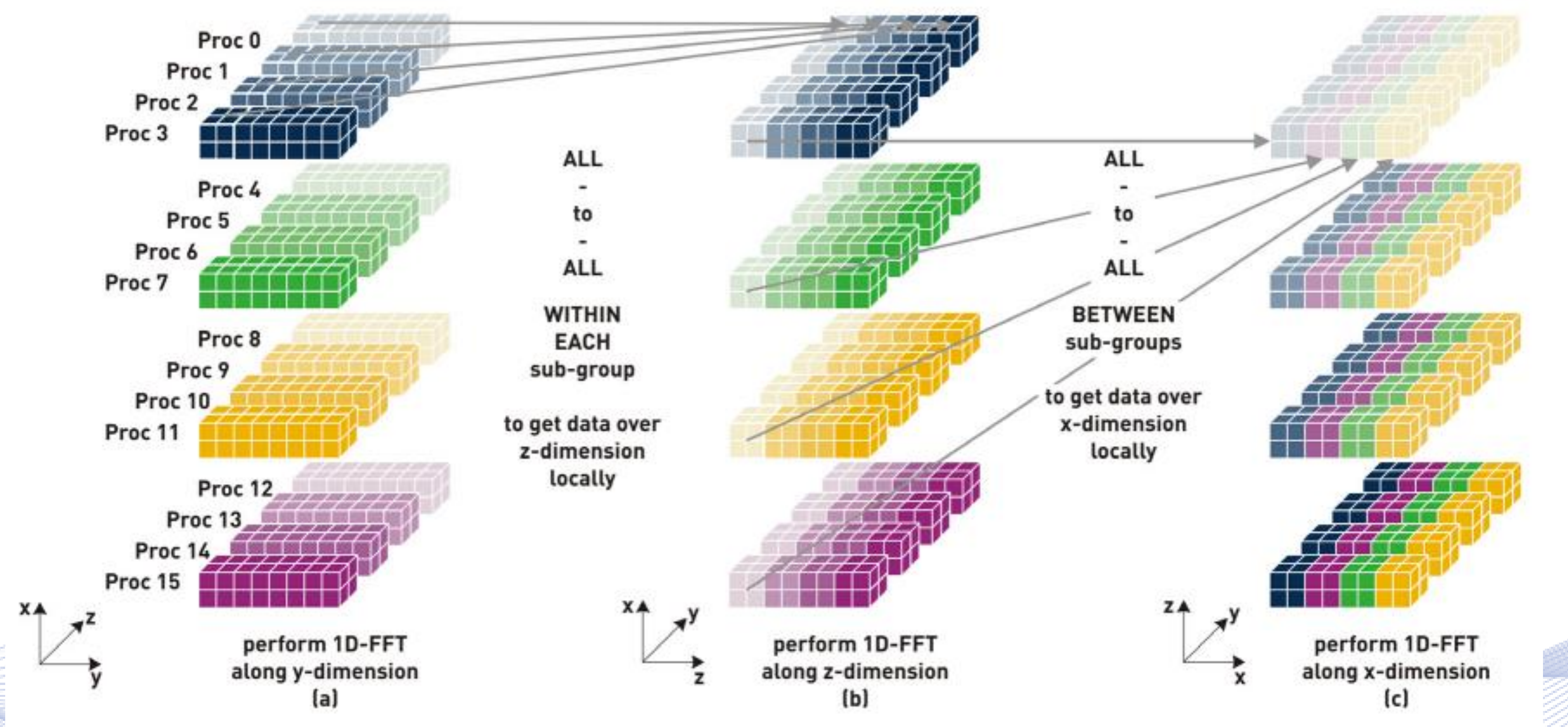
$$y(n_1, n_2, n_3) = x(n_1, n_2, n_3) \circledast \circledast \circledast \, h(n_1, n_2, n_3)$$

$$\leftrightarrow Y(k_1, k_2, k_3) = X(k_1, k_2, k_3)\, H(k_1, k_2, k_3)$$

- Thus, the 3D cyclic convolution can be computed by DFT:



# 3D Fast Fourier Transform



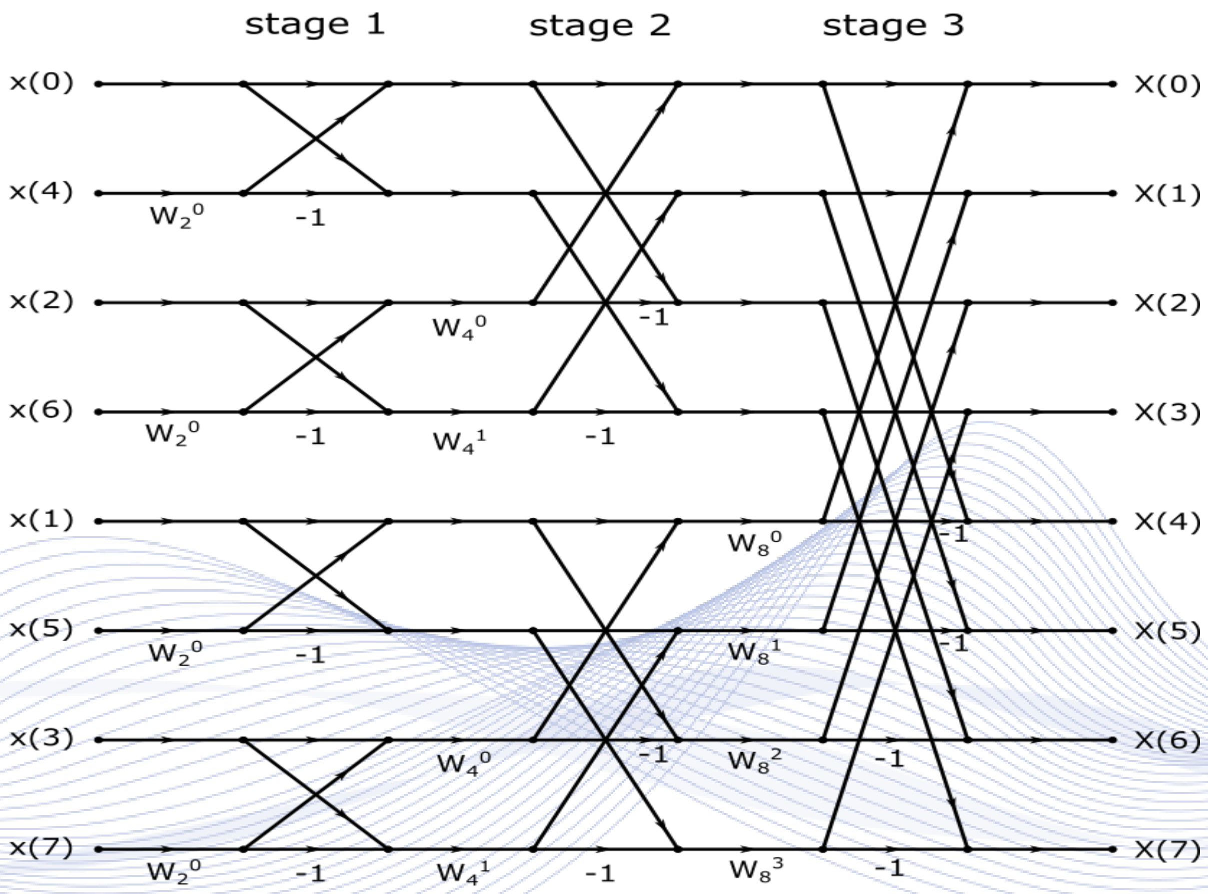

Decomposition of 3D FFT into 1D FFTs (from [HEI2005]).

# 3D Fast Fourier Transform

- There are many variants of 1D FFT algorithms. The best known is the Cooley-Tuckey ***radix-2 decimation in time*** (DIT) FFT algorithm.

- It uses the "divide and conquer" approach by recursively breaking down the 1D DFT of any composite size $N = N_1 N_2$ into $N_1$ smaller DFTs of sizes $N_2$.

# 3D Fast Fourier Transform




Data flow diagram of 1D radix-2 FFT algorithm for $N$=8.

Artificial Intelligence & Information Analysis Lab

# 3D Fast Fourier Transform

The number of additions and multiplications required for computing the 3D FFT by using 1D radix-2 FFTs is [PIT2000]:

$$A = N_1 N_2 N_3 \log_2(N_1 N_2 N_3)$$

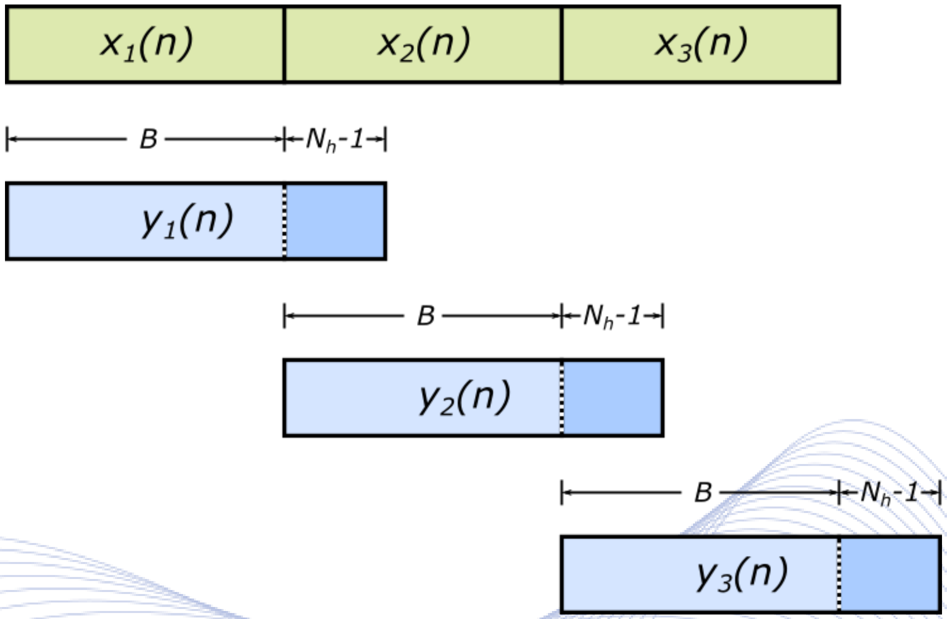
$$M = \frac{N_1 N_2 N_3}{2} \log_2(N_1 N_2 N_3)$$

This is much better as compared to $(N_1 N_2 N_3)^2$ multiplications required for the direct computation of 3D DFT.

# Block convolutions

- Computation of convolution by DFT methods (using FFT algorithms) for signals of large sizes can be very memory consuming.
- To overcome this problem, block methods can be used.
- Limiting the size of blocks limits the amount of storage required while maintaining the efficiency of the procedure [DUD1984].
- There are two block-based methods: ***overlap-add*** and ***overlap-save***.

# Overlap-add method




Overlap-add method for convolution in 1D. The input signal $x(n)$ is partitioned into three blocks $x_1(n)$, $x_2(n)$ and $x_3(n)$, each of length $B$. The impulse response $h(n)$ is of length $N_h$ and the output blocks $y_i(n) = (x_i * h)(n)$, $i = 1, 2, 3$, are of length $B + N_h - 1$ each. There are $N_h - 1$ overlapping points between output blocks $y_i(n)$ and $y_{i+1}(n)$. The output signal $y(n)$ is formed by adding all the overlapping output blocks $y_i(n)$.

# Overlap-save method

- The overlap-save method is an alternative block method.
- The 3D output is partitioned into $B_1 \times B_2 \times B_3$ non-overlapping blocks:

$$y(n_1, n_2, n_3) = \sum_i \sum_j \sum_k y_{ijk}(n_1, n_2, n_3)$$

- The corresponding 3D input section $x_{ijk}(n_1, n_2, n_3)$ of size $B_1 \times B_2 \times B_3$ is extended to $x'_{ijk}(n_1, n_2, n_3)$ of size $(B_1 + N_{h_1} - 1) \times (B_2 + N_{h_2} - 1) \times (B_3 + N_{h_3} - 1)$.

# Overlap-save method

- The 3D cyclic convolution $y'_{ijk} = x'_{ijk} \circledast \circledast \circledast h$ can be efficiently computed by FFT of size $(B_1 + N_{h_1} - 1) \times (B_2 + N_{h_2} - 1) \times (B_3 + N_{h_3} - 1)$.
- Each of the resulting blocks $y'_{ijk}$ will contain a sub-block of size $B_1 \times B_2 \times B_3$ which is identical to the desired linear convolution $y_{ijk} = x_{ijk} *** h$ (which are added to form $y$).
- In both block methods the choice of block size affects the amount of storage needed and the amount of computations.

# Winograd convolution algorithm

- We saw that the 3D cyclic convolution can be efficiently computed by 1D FFTs.
- When the length of the convolution kernel is small, the best convolution algorithms, as measured by the number of required multiplications, are the Winograd convolution algorithms [BLA2010].
- The Winograd convolution algorithms are based on the ***Chinese Remainder Theorem (CRT)*** for polynomials.

# 1D Winograd convolution

For simplicity, we first present the 1D Winograd convolution algorithm and later extend it to 3D.

The 1D cyclic convolution of length $N$ can be expressed in terms of polynomials in z-domain as:

$$Y(z) = X(z)H(z) \mod z^N - 1$$

# 1D Winograd convolution

- The Winograd convolution algorithm can be expressed compactly in the following matrix notation (bilinear form):

$$\mathbf{y} = \mathbf{C}(\mathbf{Ax} \otimes \mathbf{Bh})$$

  where $\odot$ denotes element-wise product.
- Matrices $\mathbf{A}$ and $\mathbf{B}$ typically have elements $-1, 0, 1$. Therefore products $\mathbf{Ax}$ and $\mathbf{Bh}$ represent additions instead of multiplications.

# 3D Winograd convolution

- The Winograd convolution can be extended to three dimensions [PIT1987].
- The 3D cyclic convolution can be expressed as:

$Y(z_1, z_2, z_3) = X(z_1, z_2, z_3)\, H(z_1, z_2, z_3)$ mod $P_1(z_1), P_2(z_2), P_3(z_3)$,

where $P_i(z_i) = \left(z_i^{N_i} - 1\right),\ i = 1, 2, 3.$

- Each $P_i(z_i)$ can be factorized into $\nu_i$ cyclotomic polynomials:

$$P_i(z_i) = \prod_{j_i=1}^{\nu_i} p_{ij_i}(z_i), \qquad \deg\{p_{ij_i}\} = N_{ij_i},\ i = 1,2,3$$

# 3D Winograd convolution

- The number of multiplications of this 3D algorithm is:

$$(2N_1 - \nu_1)(2N_2 - \nu_2)(2N_3 - \nu_3),$$

  i.e., the computational complexity is of order $O(N^3)$.

- However, this is not the minimal computational complexity, because there can exist further factorizations such as each $p_{1j_1}(z_1)$ over $Q[z_2]/p_{2j_2}(z_2)$ or $Q[z_3]/p_{3j_3}(z_3)$ etc.

# 3D Winograd convolution

It can be shown that the optimal algorithm for 3D cyclic convolution exists and requires the following minimum number of multiplications [PIT1987]:

$$M = \sum_{j_1=1}^{\nu_1} \sum_{j_2=1}^{\nu_2} \sum_{j_3=1}^{\nu_3} M_{j_1 j_2 j_3}$$

where

$$\begin{aligned} M_{j_1 j_2 j_3} = \min \{ & (2N_{1j_1} - 1)(2N_{2j_2} - k_{2j_2})(2N_{3j_3} - k_{3j_3}), \\ & (2N_{2j_2} - 1)(2N_{1j_1} - k_{1j_1})(2N_{3j_3} - k_{3j_3}), \\ & (2N_{3j_3} - 1)(2N_{1j_1} - k_{1j_1})(2N_{2j_2} - k_{2j_2}) \} \end{aligned}$$

# 3D Winograd convolution

- Such optimal algorithms can be expressed in the matrix form that we saw for 1D Winograd convolution:

$$\mathbf{y} = \mathbf{R}\mathbf{B}^{\mathrm{T}}\left(\mathbf{A}\mathbf{x} \otimes \mathbf{C}^{\mathrm{T}}\mathbf{R}\mathbf{h}\right)$$

  which then can be computed by using linear algebra libraries such as BLAS and cuBLAS.

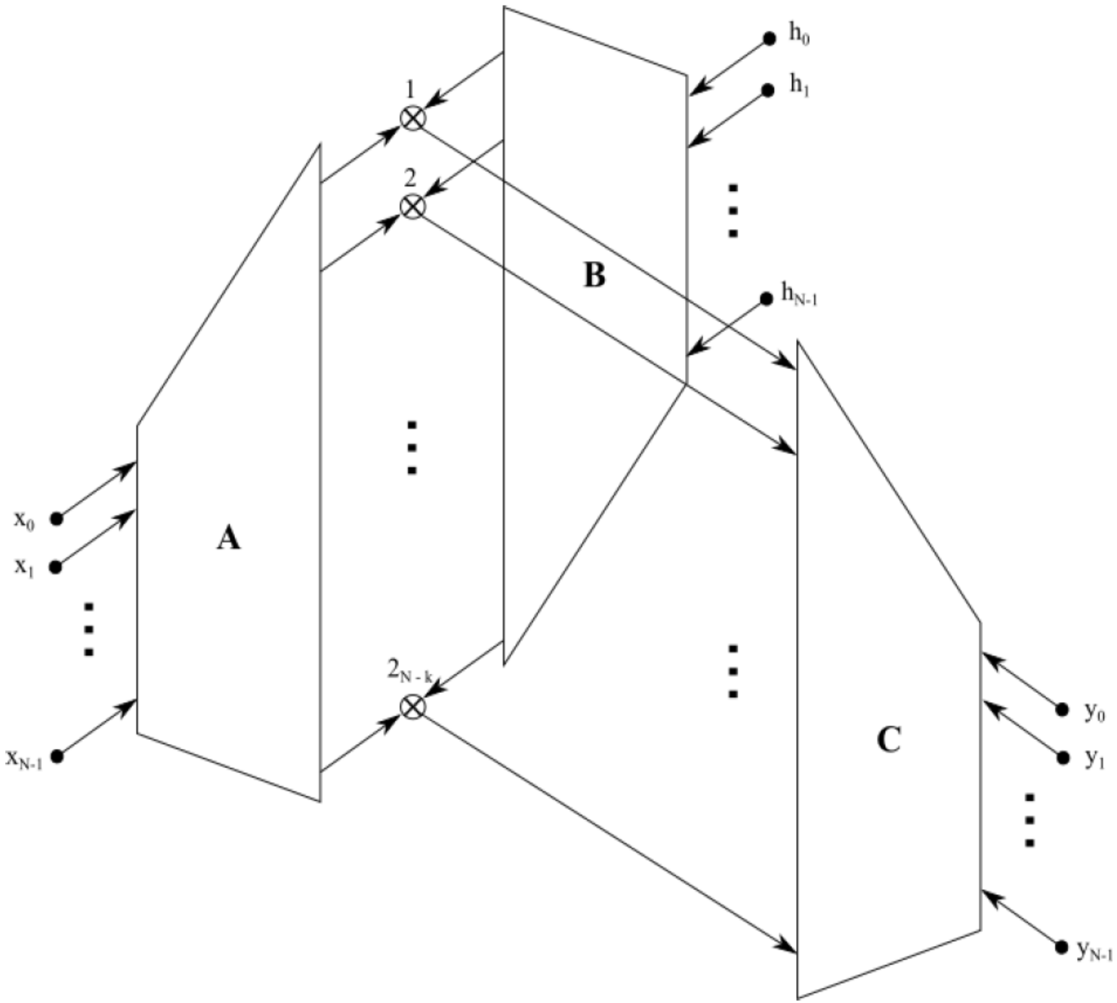
- However, finding the matrices A, B, C can be a tedious task and has to be done by hand for a desired convolution size.

# Winograd 3D cyclic convolutions

- Winograd 3D convolution algorithms or fast 3D filtering:

$$\mathbf{y} = \mathbf{C}(\mathbf{Ax} \otimes \mathbf{Bh}).$$

- GEneral Matrix Multiplication (GEMM) BLAS or cuBLAS routines can be used.



# Bibliography

[PIT2017] I. Pitas, “Digital video processing and analysis” , China Machine Press, 2017 (in Chinese).

[PIT2013] I. Pitas, “Digital Video and Television” , Createspace/Amazon, 2013.

[PIT2021] I. Pitas, “Computer vision”, Createspace/Amazon, in press.

[NIK2000] N. Nikolaidis and I. Pitas, “3D Image Processing Algorithms”, J. Wiley, 2000.

[PIT2000] I. Pitas, “Digital Image Processing Algorithms and Applications”, J. Wiley, 2000.

[PIT1987] I. Pitas, M. Strintzis, “Multidimensional cyclic convolution algorithms with minimal multiplicative complexity”, IEEE transactions on acoustics, speech, and signal processing, vol. 35, no. 3, pp. 384-390, 1987.

# Q & A

Thank you very much for your attention!

More material in
http://icarus.csd.auth.gr/cvml-web-lecture-series/

Contact: Prof. I. Pitas
pitas@csd.auth.gr