User manual (MATLAB\_CUDA\_MEX)

/\*

\* Discrete Cosine/Sine Transform (DCT/DST and IDCT/IDST one to four-all in one)

\* DCT/DST and IDCT/IDST I ---> IV

\* This CUDA code can handle/work with any type of the input mxArrays,

\* GPUarray or standard matlab CPU array as input {prhs [0]:= mxGPUArray or CPU Array}

\* GpuArray/cpuArray output, B=Discrete Transform (A, type of Transform (sine or cosine), type of Transform (direct/inverse), type of DCT/DST or IDCT/IDST, dimensions).

\* Developed at UCL, Institute of Neurology, 12 Queen Square, WC1N 3AR, London

\* Welcome Trust Centre for Neuroimaging

\* Part of the project SPM (http://www.fil.ion.ucl.ac.uk/spm)

\* Copyright 2018

\* Kevin Bronik

\*/

To compile:

First try the method described here:

<https://uk.mathworks.com/help/distcomp/run-mex-functions-containing-cuda-code.html>

After successful compiling running and testing then simply try following statement (copy and paste in Matlab and enter):

>> debug\_Discrete\_Transform(false)

See the file “debug\_Discrete\_Transform.m”

## **To compute Discrete Cosine/Sine Transform DCT/DST and inverse Discrete Cosine/Sine Transform IDCT/IDST user can choose/use the following unified syntax:**

**B=Discrete\_Transform** (Input array A, Type of Discrete Transform, Type of Transformation, Type of DCT/DST or IDCT/IDST, Dimensions)

Where

B: = **output array same type as input array**

Input array A: = **array, or gpuArray object**

Type of Discrete Transform: = **sine or cosine**

Type of Transformation: = **direct or inverse**

Type of DCT/DST or IDCT/IDST: = **one, two, three or four** **(I, II, III, IV)**

### Dimensions: = **row or column** (**dimension to operate along**)

**Examples:**

**(First example**)

>> a = [1, 2, 3; 4, 5, 6; 7, 8, 9]; 🡨--- (original input array)

>> a=single (a)

a =

3×3 single matrix

1 2 3

4 5 6

7 8 9

>> e=Discrete\_Transform (a, 'cosine', 'direct', 'two', 'row')

🡨--- (direct transform)

e =

3×3 single matrix

6.9282 8.6603 10.3923

-4.2426 -4.2426 -4.2426

-0.0000 -0.0000 -0.0000

>> d=Discrete\_Transform (e, 'cosine', 'inverse', 'two', 'row')

🡨--- (inverse transform-recovery)

d =

3×3 single matrix

1.0000 2.0000 3.0000

4.0000 5.0000 6.0000

7.0000 8.0000 9.0000

>>

**(Second example**)

>> a = single (ones (12, 5,'gpuArray')); 🡨--- (original input array)

>> a

a =

12×5 single gpuArray matrix

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

>> e=Discrete\_Transform (a, 'sine', 'direct', 'three', 'column')

🡨--- (direct transform)

e =

12×5 single gpuArray matrix

3.1277 1.0668 0.6706 -0.0000 0.0000

3.1277 1.0668 -0.0000 -0.0000 0.0000

3.1277 1.0668 -0.0000 -0.0000 0.4118

3.1277 0.0000 -0.0000 -0.0000 0.4118

3.1277 0.0000 -0.0000 0.4419 0.4118

0 0.0000 -0.0000 0.4419 0.4118

0 0.0000 0.5146 0.4419 0.4118

0 0.0000 0.5146 0.4419 0

0 0.6706 0.5146 0.4419 0

0 0.6706 0.5146 0.0000 0

1.0668 0.6706 0.5146 0.0000 0

1.0668 0.6706 -0.0000 0.0000 0

>> f=Discrete\_Transform (e, 'sine', 'inverse', 'three', 'column')

🡨--- (inverse transform-recovery)

f =

12×5 single gpuArray matrix

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

>>

**(Third example**)

Working alongside existing MATLAB dct

Modified version of the following matlab example.

openExample('signal/ImageResizingExample')

%% Image Resizing

% Load a file that contains depth measurements of a mold used to mint a

% United States penny. The data, taken at the National Institute of

% Standards and Technology, are sampled on a 128-by-128 grid. Display the

% data.

load penny

surf(P)

view(2)

colormap copper

shading interp

axis ij square off

%%

% Compute the discrete cosine transform of the image data using the DCT-1

% variant. Operate first along the rows and then along the columns.

%Discrete\_Transform(e, 'cosine', 'inverse', 'four' , 'row')

%Q = dct(P,[],1,'Type',1); 🡨--------- original matlab dct

%R = dct(Q,[],2,'Type',1); 🡨--------- original matlab dct

%

Q = Discrete\_Transform (single (P), 'cosine', 'direct', ‘one’, 'row');

R = Discrete\_Transform (single (Q), 'cosine', 'direct', ‘one’, 'column');

🡨-- Replaced with the new syntax

%%

% Invert the transform. Truncate the inverse so that each dimension of the

% reconstructed image is one-half the length of the original.

S = idct(R,size(P,2)/2,2,'Type',1);

T = idct(S,size(P,1)/2,1,'Type',1);

%%

% Invert the transform again. Zero-pad the inverse so that each dimension

% of the reconstructed image is twice the length of the original.

U = idct(R,size(P,2)\*2,2,'Type',1);

V = idct(U,size(P,1)\*2,1,'Type',1);

%%

% Display the original and reconstructed images.

surf(V)

view(2)

shading interp

hold on

surf(P)

view(2)

shading interp

surf(T)

view(2)

shading interp

hold off

axis ij equal off

## **To compute two dimensional (full 2D) discrete transform the following unified syntax can be used by user:**

**Outputdata =Discrete\_Transform\_2D** (inputdata, type1, type2, type3)

Where

**Outputdata**: = **output array same type as input array**

Inputdata: = **array, or gpuArray object “2D”**

type1: = **sine or cosine**

type2: = **direct or inverse**

type3: = **one, two, three or four** **(I, II, III, IV)**

**(First example)**

>> a = single (ones (3, 7,'gpuArray')); 🡨--- (original 2d input array)

>> a

a =

3×7 single gpuArray matrix

1 1 1 1 1 1 1

1 1 1 1 1 1 1

1 1 1 1 1 1 1

>> c=Discrete\_Transform\_2D (a, 'sine', 'direct', 'four')

🡨--- (direct transform)

c =

3×7 single gpuArray matrix

3.7652 0.5962 0.4242 0.2900 0.1635 0.3420 0.1334

1.2764 0.4979 1.3781 0.2182 0.1553 0.2123 0.1197

0.7924 0.4466 0.4672 0.1822 1.0089 0.1597 0.1137

>> d=Discrete\_Transform\_2D(c, 'sine', 'inverse', 'four')

🡨--- (inverse transform-recovery)

d =

3×7 single gpuArray matrix

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

>>

**(Second example)**

>> a = single (ones (4, 6)); 🡨--- (original 2d input array)

>> a

a =

4×6 single matrix

1 1 1 1 1 1

1 1 1 1 1 1

1 1 1 1 1 1

1 1 1 1 1 1

>> c=Discrete\_Transform\_2D (a, 'cosine', 'direct', 'three')

🡨--- (direct transform)

c =

4×6 single matrix

4.6246 0.4783 -0.0000 0.0000 0.0000 -0.0000

-1.1547 0.1631 0.0000 -0.0000 0.0000 0.0000

0.9916 -0.0000 -0.0000 0.0000 -0.0000 -0.0000

-0.2039 0.0000 -0.0000 -0.0000 0.0000 -0.0000

>> d=Discrete\_Transform\_2D (c, 'cosine', 'inverse', 'three')

🡨--- (inverse transform-recovery)

d =

4×6 single matrix

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

>>

## **To compute three dimensional (3D) discrete transform the following unified syntax can be used by user:**

[A modificated version of the following originally introduced algorithm (in MATLAB)

<https://uk.mathworks.com/matlabcentral/fileexchange/52494-3d-image-watermarking-using-dwt--dct--dct+dwt?focused=3892547&tab=function> ]

**Outputdata =Discrete\_Transform\_3D** (inputdata, type1, type2, type3, DIM)

Where

**Outputdata**: = **output array same type as input array**

Inputdata: = **array, or gpuArray object “3D”**

type1: = **sine or cosine**

type2: = **direct or inverse**

type3: = **one, two, three or four** **(I, II, III, IV)**

DIM: = row, column, third or full

(**Dimension to operate along**)

row (first dimension)

column (second dimension)

third (third dimension)

full (to do full transform)

**Examples:**

**(First example)**

>> a = single (ones (9, 5, 2)); 🡨--- (original 3d input array)

>> a

9×5×2 single array

a(:,:,1) =

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

a(:,:,2) =

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

>> M=Discrete\_Transform\_3D (a, 'cosine', 'direct', 'two', 'full')

🡨--- (direct transform)

9×5×2 single array

M(:,:,1) =

8.8282 -0.0000 0.0000 -0.0000 0

0 1.9072 -0.0000 0.0000 0.0000

0.0000 0 1.0541 -0.0000 -0.0000

-0.0000 0.0000 0 0.6563 0.0000

0.0000 -0.0000 0.0000 0 0.3739

-2.4730 0.0000 -0.0000 0.0000 0

0 -0.7557 0.0000 -0.0000 0.0000

-0.0000 0 -0.2132 0.0000 -0.0000

0.0000 -0.0000 0 0.1090 0.0000

M(:,:,2) =

1.0e-06 \*

0.1436 -0.0000 -0.0000 0.0000 0

0 -0.0078 0.0000 -0.0000 -0.0000

-0.0000 0 -0.0160 0.0000 -0.0000

-0.0000 0.0000 0 -0.0128 0.0000

-0.0000 -0.0000 -0.0000 0 0.0063

-0.0523 0.0000 0.0000 0.0000 0

0 -0.0022 0.0000 0.0000 0.0000

-0.0000 0 -0.0021 -0.0000 -0.0000

0.0000 0.0000 0 0.0005 -0.0000

>> N= Discrete\_Transform\_3D (M, 'cosine', 'inverse', 'two', 'full')

🡨--- (inverse transform-recovery)

9×5×2 single array

N(:,:,1) =

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

N(:,:,2) =

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

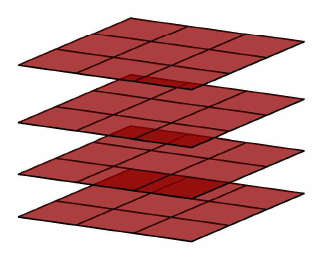
1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000

……………………………………………………………………………………………………………………………………………………………..



**(Second example**) **Row wise discrete transform**

>> a = single (ones (5, 9, 4,'gpuArray')); 🡨--- (original 3d input array)

>> a

a(:,:,1) =

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

a(:,:,2) =

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

a(:,:,3) =

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

a(:,:,4) =

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

>> b= Discrete\_Transform\_3D (a, 'sine', 'direct', 'one', 'row')

🡨--- (direct transform)

b(:,:,1) =

5.4961 0.8705 0.1379 0 -0.0000 0.4033 0.1047 -0.0000 0.0000

5.4961 0.8705 0.1379 0 -0.0000 0.4033 0.1047 -0.0000 0.0000

5.4961 0.8705 0.1379 0 -0.0000 0.4033 0.1047 -0.0000 0.0000

5.4961 0.8705 0.1379 0 -0.0000 0.4033 0.1047 -0.0000 0.0000

5.4961 0.8705 0.1379 0 -0.0000 0.4033 0.1047 -0.0000 0.0000

b(:,:,2) =

1.0e-15 \*

-0.3242 0.4322 -0.1876 -0.0152 0.0004 0.0510 0.1530 0.1518 0.0362

-0.3242 0.4322 -0.1876 -0.0152 0.0004 0.0510 0.1530 0.1518 0.0362

-0.3242 0.4322 -0.1876 -0.0152 0.0004 0.0510 0.1530 0.1518 0.0362

-0.3242 0.4322 -0.1876 -0.0152 0.0004 0.0510 0.1530 0.1518 0.0362

-0.3242 0.4322 -0.1876 -0.0152 0.0004 0.0510 0.1530 0.1518 0.0362

b(:,:,3) =

1.7085 0.4435 0.0000 0 1.2975 0.2055 0.0325 0.0000 0.0000

1.7085 0.4435 0.0000 0 1.2975 0.2055 0.0325 0.0000 0.0000

1.7085 0.4435 0.0000 0 1.2975 0.2055 0.0325 0.0000 0.0000

1.7085 0.4435 0.0000 0 1.2975 0.2055 0.0325 0.0000 0.0000

1.7085 0.4435 0.0000 0 1.2975 0.2055 0.0325 0.0000 0.0000

b(:,:,4) =

1.0e-15 \*

0.2161 0.6483 -0.0021 -0.0295 -0.0765 0.1020 0.2829 0.0631 0.0132

0.2161 0.6483 -0.0021 -0.0295 -0.0765 0.1020 0.2829 0.0631 0.0132

0.2161 0.6483 -0.0021 -0.0295 -0.0765 0.1020 0.2829 0.0631 0.0132

0.2161 0.6483 -0.0021 -0.0295 -0.0765 0.1020 0.2829 0.0631 0.0132

0.2161 0.6483 -0.0021 -0.0295 -0.0765 0.1020 0.2829 0.0631 0.0132

>> c= Discrete\_Transform\_3D (b, 'sine', 'inverse', 'one', 'row')

🡨--- (inverse transform-recovery)

c(:,:,1) =

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

c(:,:,2) =

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

c(:,:,3) =

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

c(:,:,4) =

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

>>

**(Third example**)

3D image processing (415x312x3 image dimensions)

**3D image processing using 3D-DCT/DST**

This example shows how to process an colour image using the three-dimensional discrete cosine/sine transferm (3D DCT/DST).

Read an image into the workspace, then convert the image to double (after that, convert it to Single-precision floating).

[X,map] = imread('corn.tif');

if ~isempty(map)

Im = ind2rgb(X,map);

end

Perform a 3D DCT of the colour image using the Discrete\_Transform\_3D function. Working only with Single-precision

J = Discrete\_Transform\_3D(single(Im), 'cosine', 'direct', 'two', 'row');

f = Discrete\_Transform\_3D(J, 'cosine', 'direct', 'two', 'column');

Reconstruct the image using the inverse 3D DCT function Discrete\_Transform\_3D.

K = Discrete\_Transform\_3D(f, 'cosine', 'inverse', 'two', 'column');

K = Discrete\_Transform\_3D(K, 'cosine', 'inverse', 'two', 'row');

Perform a full 3D DST of the colour image using the Discrete\_Transform\_3D function, and reconstruct the image using the inverse full 3D DST function Discrete\_Transform\_3D.

full = Discrete\_Transform\_3D(single(Im), 'sine', 'direct', 'four', 'full');

back = Discrete\_Transform\_3D(full, 'sine', 'inverse', 'four', 'full');

Display the original colour image alongside the processed image.

figure;

subplot(3,2,1),imshow(Im);

title('Original Colour Image');

subplot(3,2,2),imshow(J);

title('Discrete Transform (cosine row)');

subplot(3,2,3),imshow(f);

title('Discrete Transform (cosine column)');

subplot(3,2,4),imshow(K);

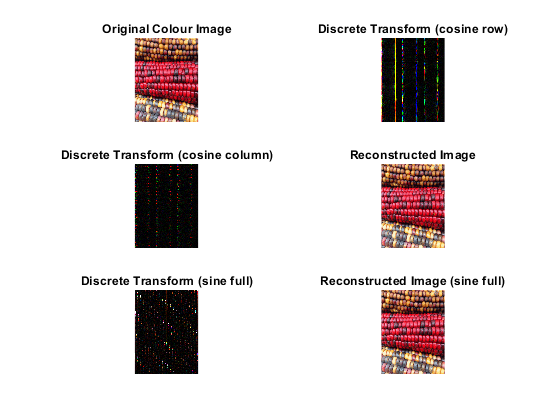
title('Reconstructed Image');

subplot(3,2,5),imshow(full);

title('Discrete Transform (sine full)');

subplot(3,2,6),imshow(back);

title('Reconstructed Image (sine full)');



GPUarray version of sine transform

**3D image processing using 3D-DST and GPUarray**

This example shows how to process an colour image using the three-dimensional discrete sine transform (3D DST).

Read an image into the workspace, then convert the image to double.

[X, map] = imread ('corn.tif');

if ~isempty (map)

Im = ind2rgb(X, map);

end

Perform a 3D DST of the colour image using the Discrete\_Transform\_3D function.

Im=gpuArray (Im);

J = Discrete\_Transform\_3D (single (Im), 'sine', 'direct', 'one', 'full');

Reconstruct the image using the inverse 3D DST function Discrete\_Transform\_3D.

K = Discrete\_Transform\_3D (single (J), 'sine', 'inverse', 'one', 'full');

Display the original colour image alongside the processed image.

K=gather (K);

Im=gather (Im);

figure

imshowpair(Im, K,'montage')

title('Original Colour Image (Left) and reconstructed

Image (Right)'); 

Note that, to get maximum performance user can try first to define all input arrays as 'gpuArray' before running the Discrete Transform.

Note also that, the above syntax can be easily extended to higher dimensions (n dimensional Discrete Transform).