User manual (**Element wise Multiplication**)

/\*

\* Element wise Multiplication

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

\* GPUarray or standard matlab CPU array as input {prhs [0], prhs [1]:= mxGPUArray or CPU Array}

\* GpuArray output, C=ELM\_CUDA (A, B) or C3=ELM\_CUDA\_3D (A3, B3)

\* 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

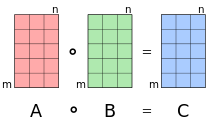
\* Kevin Bronik

\*/

Replacing Matlab operator “.\*” with parallel Matlab-CUDA syntax

Hadamard product (matrices)

**Reference:** From Wikipedia, the free encyclopaedia

[](https://en.wikipedia.org/wiki/File:Hadamard_product_qtl1.svg)

The Hadamard product operates on identically-shaped matrices and produces a third matrix of the same dimensions.

In [mathematics](https://en.wikipedia.org/wiki/Mathematics), the **Hadamard product** (also known as the **Schur product**[[1]](https://en.wikipedia.org/wiki/Hadamard_product_(matrices)#cite_note-1) or the **entry wise product** is a [binary operation](https://en.wikipedia.org/wiki/Binary_operation) that takes two [matrices](https://en.wikipedia.org/wiki/Matrix_(mathematics)) of the same dimensions, and produces another matrix where each element *i,j* is the product of elements *i,j* of the original two matrices. It should not be confused with the more common [matrix product](https://en.wikipedia.org/wiki/Matrix_multiplication). It is attributed to, and named after, either French mathematician [Jacques Hadamard](https://en.wikipedia.org/wiki/Jacques_Hadamard), or German mathematician [Issai Schur](https://en.wikipedia.org/wiki/Issai_Schur).

The Hadamard product is [associative](https://en.wikipedia.org/wiki/Associative) and [distributive](https://en.wikipedia.org/wiki/Distributive_property), and unlike the matrix product it is also [commutative](https://en.wikipedia.org/wiki/Commutative).

**Examples:**

**(First example**)

**3D image processing using element wise multiplication**

This example shows how to process two colour images using the three-dimensional Element wise Multiplication

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

[X,map] = imread('photo.tiff');

if ~isempty(map)

Im1 = ind2rgb(X,map);

end

[Y,map] = imread('gibson\_robotlp\_red.tiff');

if ~isempty(map)

Im2 = ind2rgb(Y,map);

end

Perform a 3D Element wise Multiplication of the colour images using the ELM\_CUDA\_3D function.

K = ELM\_CUDA\_3D (Im1, Im2);

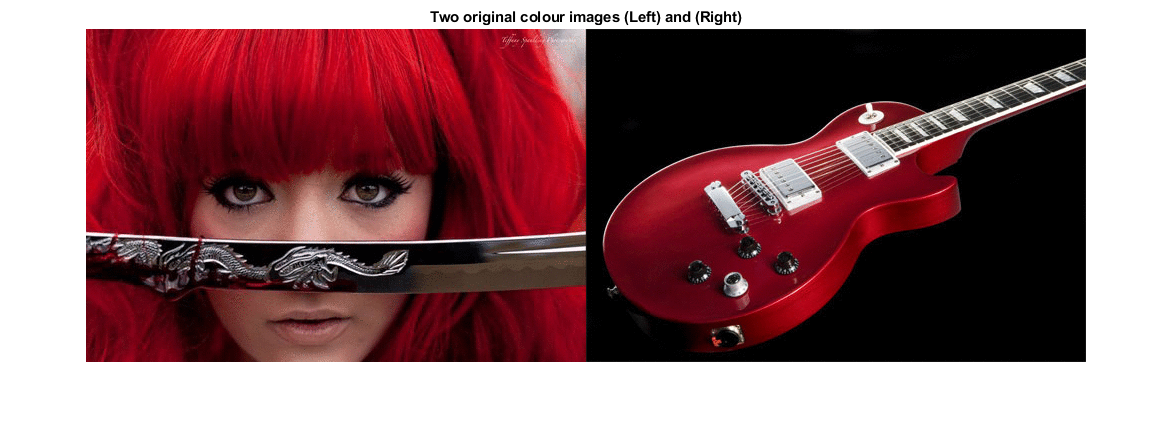
Display the original colour image alongside the processed image.

figure

imshowpair(Im1,Im2,'montage')

title('Two original colour images (Left) and (Right)');

Before Processing:



>> whos Im1, Im2

Name Size Bytes Class Attributes

Im1, Im2 333x500x3 (Image Dimensions) 3996000 double

After Processing [K = ELM\_CUDA\_3D (Im1, Im2)]:

figure

imshowpair(Im1,K,'montage')

title('Original colour image (Left) and processed image (Right)');



**(Second example**)

>> A 🡨--- (3D input array)

A(:,:,1) =

1 2 3

4 5 6

7 8 9

A(:,:,2) =

10 11 12

13 14 15

16 17 18

A(:,:,3) =

19 20 21

22 23 24

25 26 27

>> B 🡨--- (3D input array)

B(:,:,1) =

1 2 3

4 5 6

7 8 9

B(:,:,2) =

10 11 12

13 14 15

16 17 18

B(:,:,3) =

19 20 21

22 23 24

25 26 27

>> A=gpuArray (A) 🡨--- (To get maximum performance)

>> B=gpuArray (B)

>> C=**ELM\_CUDA\_3D** (A, B) 🡨--- (3D output array-gpuArray)

C(:,:,1) =

1 4 9

16 25 36

49 64 81

C(:,:,2) =

100 121 144

169 196 225

256 289 324

C(:,:,3) =

361 400 441

484 529 576

625 676 729

>>

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\_ELM\_CUDA\_cu(false)

See the file “debug\_ELM\_CUDA\_cu.m”

{\displaystyle \left({\begin{array}{ccc}\mathrm {a} \_{11}&\mathrm {a} \_{12}&\mathrm {a} \_{13}\\\mathrm {a} \_{21}&\mathrm {a} \_{22}&\mathrm {a} \_{23}\\\mathrm {a} \_{31}&\mathrm {a} \_{32}&\mathrm {a} \_{33}\end{array}}\right)\circ \left({\begin{array}{ccc}\mathrm {b} \_{11}&\mathrm {b} \_{12}&\mathrm {b} \_{13}\\\mathrm {b} \_{21}&\mathrm {b} \_{22}&\mathrm {b} \_{23}\\\mathrm {b} \_{31}&\mathrm {b} \_{32}&\mathrm {b} \_{33}\end{array}}\right)=\left({\begin{array}{ccc}\mathrm {a} \_{11}\,\mathrm {b} \_{11}&\mathrm {a} \_{12}\,\mathrm {b} \_{12}&\mathrm {a} \_{13}\,\mathrm {b} \_{13}\\\mathrm {a} \_{21}\,\mathrm {b} \_{21}&\mathrm {a} \_{22}\,\mathrm {b} \_{22}&\mathrm {a} \_{23}\,\mathrm {b} \_{23}\\\mathrm {a} \_{31}\,\mathrm {b} \_{31}&\mathrm {a} \_{32}\,\mathrm {b} \_{32}&\mathrm {a} \_{33}\,\mathrm {b} \_{33}\end{array}}\right)}