Blur Test

1.0.0

Generated by Doxygen 1.9.6

Chapter 1

Blur-Test

Testing out some blurs with opecv, OpenMP and CUDA

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

File Documentation

3.1 blur.cu File Reference

convolution blurring in Nvidia CUDA

```
#include <cuda_runtime.h>
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/core/cuda.hpp>
#include <opencv2/core/cuda/common.hpp>
#include <opencv2/core/matx.hpp>
#include <opencv2/cudaimgproc.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/opencv.hpp>
#include <stdio.h>
```

Macros

- #define SAFE_CALL(call, msg) _safe_cuda_call((call), (msg), __FILE__, __LINE__)
  a macro for sage calling CUDA functions

Functions

- __host__ void generate_gaussian_kernel_2d (float *kernel, const int n, const float sigma=1)
  generate the gaussian kernel with given kernel size and standard deviation
- __host__ void generate_gaussian_kernel_1d (float *kernel, const int n, const float sigma=1)
  generate a 1D gaussian kernel
- __device__ __forceinline__ void set_value (const int &val, uchar &out)
  Sets the value of a uchar type.
- __device__ __forceinline__ void set_value (const float &val, float &out)
  set the value for a floating point type.
- __device__ __forceinline__ void set_value (const float &val, float3 &out)
  set the value for a float3 tupe. All the 3 fields will have the value `val`
- __device__ __forceinline__ void set_value (const float3 &val, uchar3 &out)
  set the value for a unsigned char3 type with a flot3 type

- __device__ __forceinline__ void set_value (const int &val, uchar3 &out)

  Sets the value of a uchar3 type.
- __device__ __forceinline__ uchar3 subtract_value (uchar3 in1, uchar3 in2)

  Subtraction for uchar3 types.
- __device__ __forceinline__ float3 add_value (float3 in1, float3 in2)

  add two values and return it
- __device__ __forceinline__ float add_value (float in1, float in2)

  add two floating point values
- __device__ __forceinline__ uchar subtract_value (uchar in1, uchar in2)

  Subtraction for uchar types.
- __device__ __forceinline__ float3 multiply_value (const float &x, const uchar3 &y)

  multiplication for float and uchar3 types. Multiply each filed in uchar3 with the float value and return a flolat3
- __device__ __forceinline__ float3 multiply_value (const float &x, const float3 &y)

  multiplication for float and float3 types. Multiply each filed in uchar3 with the float value and return a float3
- __device__ __forceinline__ float multiply_value (const float &x, const uchar &y)

  multiplication for float and uchar4 types
- template<typename T_in , typename T_out , typename F_cal >
  __global__ void gaussian_blur (const float *kernel, int n, const cv::cuda::PtrStepSz< T_in > input, cv::cuda::PtrStepSz< T_out > output)

  applys the gaussian blur convolution to the input image
- template<typename T_in , typename T_out , typename F_cal >
  __global__ void **gaussian_blur_x** (float *kernel, int kernel_size, const cv::cuda::PtrStepSz< T_in > input, cv::cuda::PtrStepSz< T_out > output)
- template<typename T_in , typename T_out , typename F_cal >
  __global__ void **gaussian_blur_y** (float *kernel, int kernel_size, const cv::cuda::PtrStepSz< T_in > input, cv::cuda::PtrStepSz< T_out > output)
- template<class... Ts>
  void gaussian_blur_exit (Ts &&...inputs)

  free all the GPU resources
- void call_gaussian_blur_2d (float *d_kernel, const int &n, const cv::cuda::GpuMat &input, cv::cuda::GpuMat &output)

  calls the gaussian_blur function appropriately based on the type of image
- void call_gaussian_blur_1d (float *d_kernel, const int &n, const cv::cuda::GpuMat &input, cv::cuda::GpuMat &output)

  calls the separable gaussian_blur function appropriately based on the type of image
- __host__ void gaussian_blur (const cv::Mat &input, cv::Mat &output, const int n=3, const float sigma=1.0, bool two_d=true)

  the gaussian blur function which runs on the HOST CPU. It calls the `call_gaussian_blur` function after initialization of the appropriate values and kernel.
- void gaussian_blur_init (const cv::Mat &input, cv::Mat &output)

  initialization for gaussian blurring operation
- int **main** (int argc, char **argv)

Variables

- cv::cuda::GpuMat **ginput**
- cv::cuda::GpuMat **goutput**

3.1.1 Detailed Description

convolution blurring in Nvidia CUDA

Author

Arjun31415

3.1.2 Macro Definition Documentation

3.1.2.1 SAFE_CALL

```
#define SAFE_CALL(
              call,
              msg ) _safe_cuda_call((call), (msg), __FILE__, __LINE__)
```

a macro for sage calling CUDA functions

Parameters

call	the CUDA function call
msg	user specified message

3.1.3 Function Documentation

3.1.3.1 add_value() [1/2]

```
__device__ __forceinline__ float add_value (
              float in1,
              float in2 )
```

add two floating point values

Parameters

in1	value 1
in2	value 2

Returns

the sum

3.1.3.2 add_value() [2/2]

```
__device__ __forceinline__ float3 add_value (
              float3 in1,
              float3 in2 )
```

add two values and return it

Parameters

in1	input 1
in2	intput 2

Returns

returns the added value

3.1.3.3 call_gaussian_blur_1d()

```
void call_gaussian_blur_1d (
            float * d_kernel,
            const int & n,
            const cv::cuda::GpuMat & input,
            cv::cuda::GpuMat & output )
```

calls the separable gaussian_blur function appropriately based on the type of image

Parameters

d_kernel	the kernel, stored on GPU device memory
n	the size of the kernel
input	the input image stored on the GPU
output	the output image stored on the GPU

3.1.3.4 call_gaussian_blur_2d()

```
void call_gaussian_blur_2d (
            float * d_kernel,
            const int & n,
            const cv::cuda::GpuMat & input,
            cv::cuda::GpuMat & output )
```

calls the gaussian_blur function appropriately based on the type of image

Parameters

d_kernel	the kernel, stored on GPU device memory
n	the size of the kernel
input	the input image stored on the GPU
output	the output image stored on the GPU

3.1.3.5 gaussian_blur() [1/2]

```
__host__ void gaussian_blur (
            const cv::Mat & input,
            cv::Mat & output,
            const int n = 3,
            const float sigma = 1.0,
            bool two_d = true )
```

the gaussian blur function which runs on the HOST CPU. It calls the `call_gaussian_blur` function after initialization of the appropriate values and kernel.

Parameters

input	the input image stored on the CPU memory
output	the output image stored on the CPU memory
n	the size of the Gaussian kernel, defaults to 3
sigma	the standard deviation of the Gaussian kernel, defaults to 1.
two_d	whether to use the 2D gaussian blur kernel or two separable 1D gaussian blur kernels, defaults to true

3.1.3.6 gaussian_blur() [2/2]

```
template<typename T_in , typename T_out , typename F_cal >
__global__ void gaussian_blur (
            const float * kernel,
            int n,
            const cv::cuda::PtrStepSz< T_in > input,
            cv::cuda::PtrStepSz< T_out > output )
```

applys the gaussian blur convolution to the input image

Template Parameters

t_in	the type of input image, i.e uchar for black and white, uchar3 for rgb, float3 etc
t_out	the type of output image
f_cal	the type for calculating intermediate sums and products

Parameters

kernel	the kernel to apply the convolution
n	the dimension of the kernel (n x n)
input	the input image
output	the output image

3.1.3.7 gaussian_blur_exit()

```
template<class... Ts>
void gaussian_blur_exit (
            Ts &&... inputs )
```

free all the GPU resources

Template Parameters

Ts	

Parameters

inputs	varidaic list of resources

3.1.3.8 gaussian_blur_init()

```
void gaussian_blur_init (
            const cv::Mat & input,
            cv::Mat & output )
```

initialization for gaussian blurring operation

Parameters

input	input image stored on the CPU
output	output image stored on the CPU

3.1.3.9 generate_gaussian_kernel_1d()

```
__host__ void generate_gaussian_kernel_1d (
            float * kernel,
            const int n,
            const float sigma = 1 )
```

generate a 1D gaussian kernel

Parameters

kernel	the array in which the weights are stored
n	the size of the kernel. a 1D kernel of length n is needed
sigma	the standard deviation of the kernel

Returns

3.1.3.10 generate_gaussian_kernel_2d()

```
__host__ void generate_gaussian_kernel_2d (
            float * kernel,
            const int n,
            const float sigma = 1 )
```

generate the gaussian kernel with given kernel size and standard deviation

Parameters

kernel	the array in which the weights are stored
n	the size of the kernel, t.e. n x n kernel is needed
sigma	the standard deviation

3.1.3.11 multiply_value() [1/3]

```
__device__ __forceinline__ float3 multiply_value (
            const float & x,
            const float3 & y )
```

multiplication for float and float3 types. Multiply each filed in uchar3 with the float value and return a float3

Parameters

x	Input 1
y	Input 2

Returns

value after multiplication

3.1.3.12 multiply_value() [2/3]

```
__device__ __forceinline__ float multiply_value (
            const float & x,
            const uchar & y )
```

multiplication for float and uchar4 types

Parameters

x	Input 1
y	Input 2

Returns

x∗y

3.1.3.13 multiply_value() [3/3]

```
__device__ __forceinline__ float3 multiply_value (
            const float & x,
            const uchar3 & y )
```

multiplication for float and uchar3 types. Multiply each filed in uchar3 with the float value and return a flolat3

Parameters

x	Input 1
y	Input 2

Returns

value after multiplication

3.1.3.14 set_value() [1/5]

```
__device__ __forceinline__ void set_value (
            const float & val,
            float & out )
```

set the value for a floating point type.

Parameters

val	the value
out	the output

3.1.3.15 set_value() [2/5]

```
__device__ __forceinline__ void set_value (
```

```
            const float & val,
            float3 & out )
```

set the value for a float3 tupe. All the 3 fields will have the value `val`

Parameters

val	the value
out	the output

3.1.3.16 set_value() [3/5]

```
__device__ __forceinline__ void set_value (
            const float3 & val,
            uchar3 & out )
```

set the value for a unsigned char3 type with a flot3 type

Parameters

val	the value to set
out	the ouput

3.1.3.17 set_value() [4/5]

```
__device__ __forceinline__ void set_value (
            const int & val,
            uchar & out )
```

Sets the value of a uchar type.

Parameters

val	The value
out	The output

3.1.3.18 set_value() [5/5]

```
__device__ __forceinline__ void set_value (
            const int & val,
            uchar3 & out )
```

Sets the value of a uchar3 type.

Parameters

in	*val*	The value
	out	The output

3.1.3.19 subtract_value() [1/2]

```
__device__ __forceinline__ uchar subtract_value (
            uchar in1,
            uchar in2 )
```

Subtraction for uchar types.

Parameters

in	*in1*	Input 1
in	*in2*	Input 2

Returns

Output

3.1.3.20 subtract_value() [2/2]

```
__device__ __forceinline__ uchar3 subtract_value (
            uchar3 in1,
            uchar3 in2 )
```

Subtraction for uchar3 types.

Parameters

in	*in1*	Input 1
in	*in2*	Input 2

Returns

Output

3.2 main.cpp File Reference

gaussian blurring using CPU

```
#include <cmath>
#include <iostream>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/opencv.hpp>
#include <vector>
```

Functions

- void generate_gaussian_kernel (std::vector< std::vector< float > > &kernel, const int n, const float sigma=1)

  Generate a 2D gaussian kernel.
- void apply_convolution (const std ::vector< std::vector< float > > &kernel, const Mat &original_img, Mat &new_img, const int &r, const int &c)

  apply a convolution kernel to a pixel
- void apply_convolution_multi_threaded (const std::vector< std::vector< float > > &kernel, const Mat &original_img, Mat &new_img, const int &r, const int &c)

  apply a convolution kernel to a pixel using multiple threads (OMP)
- void apply_kernel (const std::vector< std::vector< float > > &kernel, const Mat &original_img, Mat &new↩ _img)

  apply a convolution kernel to the entire image
- void apply_kernel_multithreaded (const std::vector< std::vector< float > > &kernel, const Mat &original_↩ img, Mat &new_img)

  apply a convolution kernel to the entire image using multiple threads (OMP)
- int **main** (int argc, char **argv)

3.2.1 Detailed Description

gaussian blurring using CPU

Author

Arjun31415

3.2.2 Function Documentation

3.2.2.1 apply_convolution()

```
void apply_convolution (
            const std ::vector< std::vector< float > > & kernel,
            const Mat & original_img,
            Mat & new_img,
            const int & r,
            const int & c )
```

apply a convolution kernel to a pixel

Parameters

kernel	the convolution kernel
original_img	the original image
new_img	the output image
r	the row number of the current pixel
c	the column number of the current pixel

3.2.2.2 apply_convolution_multi_threaded()

```
void apply_convolution_multi_threaded (
            const std::vector< std::vector< float > > & kernel,
            const Mat & original_img,
            Mat & new_img,
            const int & r,
            const int & c )
```

apply a convolution kernel to a pixel using multiple threads (OMP)

Parameters

kernel	the convolution kernel
original_img	the original image
new_img	the output image
r	the row number of the current pixel
c	the column number of the current pixel

3.2.2.3 apply_kernel()

```
void apply_kernel (
            const std::vector< std::vector< float > > & kernel,
            const Mat & original_img,
            Mat & new_img )
```

apply a convolution kernel to the entire image

Parameters

kernel	the convolution kernel
original_img	the original image
new_img	the output image

3.2.2.4 apply_kernel_multithreaded()

```
void apply_kernel_multithreaded (
            const std::vector< std::vector< float > > & kernel,
            const Mat & original_img,
            Mat & new_img )
```

apply a convolution kernel to the entire image using multiple threads (OMP)

Parameters

kernel	the convolution kernel
original_img	the original_img
new_img	the output image

3.2.2.5 generate_gaussian_kernel()

```
void generate_gaussian_kernel (
            std::vector< std::vector< float > > & kernel,
            const int n,
            const float sigma = 1 )
```

Generate a 2D gaussian kernel.

Parameters

kernel	the kernel to be populated
n	the size of the kernel, the `kernel` must be of size $n * n$
sigma	the standard deviation of the gaussian kernel

Index