Лабораторная работа №6

Выполнил Мацур Дмитрий ст.грп ПМ-21М

1. Создание и отладка работы программы.

Требуется написать и отладить работу программы, в которой будут реализованы следующие ядра:

- Test_Kernel;
- Bilinear_Kernel;
- Gaussian_Kernel;
- Bilateral_Kernel;
- Lanczos_Kernel.

Результатом являются 7 изображений:

1) lena_gam (Test_Kernel – гамма коррекция);
2) lena_neg (Test_Kernel – негатив);
3) lena_br (Test_Kernel – преобразование яркости);
4) lena_bilinear;
5) lena_gaussian;
6) lena_bilateral;
7) lena_lanczos.

Замечание: 7 необработанных изображений формируются вручную путем копирования и соответствующего переименования файла lena.pgm.

```
#include <cuda.h>
#include <cuda_runtime.h>
#include "helper_image.h"

typedef unsigned int uint;
typedef unsigned char uchar;
texture<uchar, cudaTextureType2D, cudaReadModeElementType> g_Texture;
uint WIDTH = 512;
uint HEIGHT = 512;
```

Разделяем 1 функцию на 3

```
__global__ void Test_KernelNeg(uchar *pDst, float g, uint w, uint h)
{
  size_t tidx = threadIdx.x + blockIdx.x * blockDim.x;
  size_t tidy = threadIdx.y + blockIdx.y * blockDim.y;

  if (tidx < w && tidy < h)
  {
    float c = tex2D<uchar>(g_Texture, tidx + 0.5f, tidy + 0.5f);;
    float r = 1.0f - c;
    pDst[tidx + tidy *w] = (int)r;
  }
}
```

```
__global__ void Test_KernelGamma(uchar *pDst, float g, uint w, uint h)
{
  size_t tidx = threadIdx.x + blockIdx.x * blockDim.x;
  size_t tidy = threadIdx.y + blockIdx.y * blockDim.y;

  if (tidx < w && tidy < h)
  {
    float c = tex2D<uchar>(g_Texture, tidx + 0.5f, tidy + 0.5f);
    float r = powf(c, g);
    pDst[tidx + tidy *w] = (int)r;
  }

}

__global__ void Test_KernelBrightness(uchar *pDst, float g, uint w, ui
nt h)
{
  size_t tidx = threadIdx.x + blockIdx.x * blockDim.x;
  size_t tidy = threadIdx.y + blockIdx.y * blockDim.y;

  if (tidx < w && tidy < h)
  {
    float c = tex2D<uchar>(g_Texture, tidx + 0.5f, tidy + 0.5f);
    float r = c * 0.3f;
    pDst[tidx + tidy *w] = (int)r;
  }
}
```

В данном блоке изменений нет

```
__global__ void Gaussian_Kernel(unsigned char * pDst, float radius, fl
oat sigma_sq, int w,
                                int h)
{
  int tidx = threadIdx.x + blockIdx.x * blockDim.x;
  int tidy = threadIdx.y + blockIdx.y * blockDim.y;

  if (tidx < w && tidy < h)
  {
    float r = 0;
    float weight_sum = 0.0f;
    float weight = 0.0f;
    for (int ic = -radius; ic <= radius; ic++)
    {
      weight = exp(-(ic*ic) / sigma_sq);
      r += tex2D<uchar>(g_Texture, tidx + 0.5f + ic, tidy + 0.5f)* wei
ght;
      weight_sum += weight;
    }
    r /= weight_sum;
    pDst[tidx + tidy*w] = (int)r;
  }
}
```

```
__global__ void Bilinear_Kernel(unsigned char * dest, float factor, un
signed int w,
                                unsigned int h)
{
  size_t tidx = threadIdx.x + blockIdx.x * blockDim.x;
  size_t tidy = threadIdx.y + blockIdx.y * blockDim.y;
  // проверка, что текущие индексы не выходят за границы изображения

  if (tidx < w && tidy < h) {
    float center = tidx / factor;
    auto start = (unsigned int)center;
    unsigned int stop = start + 1.0f;
    float t = center - start;
    unsigned char a = tex2D<uchar>(g_Texture, tidy + 0.5f, start + 0.5
f);
    unsigned char b = tex2D<uchar>(g_Texture, tidy + 0.5f, stop + 0.5f
);
    float linear = (b - a)*t + a;
    dest[tidx + tidy*w] = (int)(linear);
  }
}

// Функции для загрузки и сохранения изображения
void loadImage(char *file,
               unsigned char** pixels,
               unsigned int* width,
               unsigned int* height) {
  size_t file_length = strlen(file);

  if (!strcmp(&file[file_length - 3], "pgm"))
  {
    if (!sdkLoadPGM<unsigned char>(file, pixels, width, height))
    {
      printf("Failed to load PGM image file: %s\n", file);
      exit(EXIT_FAILURE);
    }
  }
}
void saveImage(char *file, unsigned char* pixels, unsigned int width,
unsigned int height)
{
  size_t file_length = strlen(file);

  if (!strcmp(&file[file_length - 3], "pgm"))
  {
    sdkSavePGM(file, pixels, width, height);
  }
}

__device__ float Lanczos(float x, float r)
{
  const float m_pi = 3.14159265f;
  float result = 0.0f;

  if (x >= -r && x <= r) {
    float a = x*m_pi;
    float b = (r*(sin(a / r)) * (sin(a)) / (a*a));
    result = (x == 0.0f) ? 1.0f : b;
  }
  return result;
}
```

```
__global__ void Lanczos_Kernel(uchar * pDst, float factor, float blur,
float radius, float
support, float scale, uint w, uint h)
{
  size_t tidx = threadIdx.x + blockIdx.x * blockDim.x;
  size_t tidy = threadIdx.y + blockIdx.y * blockDim.y;
  // проверка, что текущие индексы не выходят за границы изображения

  if (tidx < w && tidy < h)
  {
    float r = 0;
    float weight_sum = 0.0f, weight = 0.0f;
    float center = tidx / factor;

    uint start = (uint)fmax(center - support + 0.5f, (float)0);
    uint stop = (uint)fmin(center + support + 0.5f, (float)w);

    float nmax = stop - start;
    float s = start - center;

    for (size_t n = 0; n < nmax; ++n, ++s) {
      weight = Lanczos(s*scale, radius);
      weight_sum += weight;
      r += (tex2D<uchar>(g_Texture, tidy + 0.5f, start + n + 0.5f)* we
ight);
    }
    if (weight_sum != 0.0f)
    {
      //нормализация полученных результатов
      r /= weight_sum;
    }
    pDst[tidx + tidy*w] = int(r);
  }
}
```

Вносим исправления

```
__global__ void Bilateral_Kernel(uchar * pDst, int radius, float sigma
,
                                 float weight_threshold, uint w, uint h)
{
  int area = (2 * radius + 1) * (2 * radius + 1);
  int tidx = threadIdx.x + blockIdx.x * blockDim.x;
  int tidy = threadIdx.y + blockIdx.y * blockDim.y;

  if (tidx < w && tidy < h)
  {
    float r = 0;
    float weight_sum = 0.0f;
    float weight = 0.0f;
    float weight_threshold_counter = 0.0f;
    float c00 = tex2D<uchar>(g_Texture, tidx + 0.5f, tidy + 0.5f);

    for (int ir = -radius; ir <= radius; ir++)
    {
      for (int ic = -radius; ic <= radius; ic++)
      {
```

```
float c = tex2D<uchar>(g_Texture, tidx + 0.5f + ic, tidy + 0.5f + ir);
        uint param = 0; // !!! ПОПРОБУЙТЕ ИЗМЕНЯТЬ ДАННЫЙ ПАРАМЕТР

float v1 = expf(-abs(ir * ir + ic * ic) / (2 * radius * radius));
        float v2 = expf(-abs(c - c00) *abs(c - c00) / (2 * sigma * sig
ma));

        assert(v1 < 1.01 && v2 < 1.01);

        weight = v1 * v2;
        weight_sum += weight;
        weight_threshold_counter += (weight >= weight_threshold ? 1.0f
: 0.0f);
        r += (c * weight);

      }
    }

    r /= weight_sum;
    r = c00 + (r - c00) * weight_threshold_counter / area;
    pDst[tidx + tidy * w] = (int)r;
  }
}
```

```
I.open('lena.pgm')
```

Гамма фильтр

```
I.open('lena_gamma.pgm')
```

Негатив

```
I.open('lena_neg.pgm')
```

Яркость

```
I.open('lena_brightness.pgm')
```

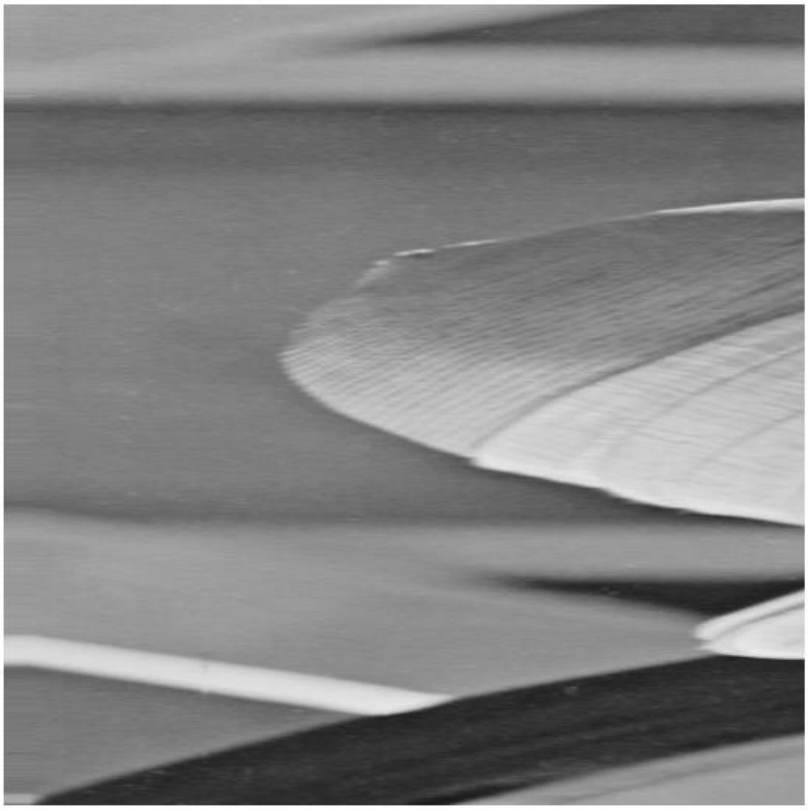
Билинейный фильтр с разными параметрами

Factor = 1.2

```
I.open('lena_bilinear.pgm')
```

Factor=4

```
I.open('lena_bilinear.pgm')
```



Гауссов филтр

```
I.open('lena_gauss.pgm')
```

Билатеральный фильтр с разными параметрами

Bilateral_Kernel<<<grid, block>>>(d_result_pixels, 3, 200, 0, WIDTH, HEIGHT);

```
I.open('lena_bilateral.pgm')
```

```
Bilateral_Kernel<<<grid, block>>>(d_result_pixels, 3, 200, 0, WIDTH, HEIGHT);
```

```
I.open('lena_bilateral.pgm')
```

Фильтр Ланцоша с разными параметрами

```
Lanczos_Kernel<<<grid, block>>> (d_result_pixels, 2, 1, 10, 1, 1, WIDTH, HEIGHT);
```

```
I.open('lena_lancroz.pgm')
```

Lanczos_Kernel<<<grid, block>>> (d_result_pixels, 1.1, 4, 10, 2, 2, WIDTH, HEIGHT);

```
I.open('lena_lancroz.pgm')
```

Поиск оптимального размера блока

Размер блока 8

```
! ../cmake-build-debug/lab6
```

```
Texture was successfully binded
Negative GPU elapsed time: 0.267872
Bilateral GPU elapsed time: 0.758944
```

Размер блока 16

Размер блока 16

```
! ../cmake-build-debug/lab6
```

```
Texture was successfully binded
Negative GPU elapsed time: 0.231392
Bilateral GPU elapsed time: 0.787552
```

Размер блока 32

```
! ../cmake-build-debug/lab6
```

```
Texture was successfully binded
Negative GPU elapsed time: 0.217792
Bilateral GPU elapsed time: 0.861824
```

Размер блока 48

```
! ../cmake-build-debug/lab6
```

```
Texture was successfully binded
Negative GPU elapsed time: 0.160384
Bilateral GPU elapsed time: 0.119104
```

Размер блока 64

```
! ../cmake-build-debug/lab6
```

```
Texture was successfully binded
Negative GPU elapsed time: 0.11856
Bilateral GPU elapsed time: 0.138944
```

Размер блока 128

```
! ../cmake-build-debug/lab6
```

```
Texture was successfully binded
Negative GPU elapsed time: 0.143936
Bilateral GPU elapsed time: 0.126016
```

Размер блока 256

```
! ../cmake-build-debug/lab6
```

```
Texture was successfully binded
Negative GPU elapsed time: 0.113312
Bilateral GPU elapsed time: 0.120352
```

Размер блока 512

```
! ../cmake-build-debug/lab6
```

```
Texture was successfully binded
Negative GPU elapsed time: 0.122176
Bilateral GPU elapsed time: 0.1032
```

Вывод:

Оптимальным размером блока будет 48, при дальнейшем росте, производительность замедляется. Замедление в данном процессе происходит из-за шины памяти. Используя данный размер блока, позволяет занять нам все потоковые процессоры устройства и при этом адресует на слишком большую область изображения.