Bilateral filtering on GPU

Contents

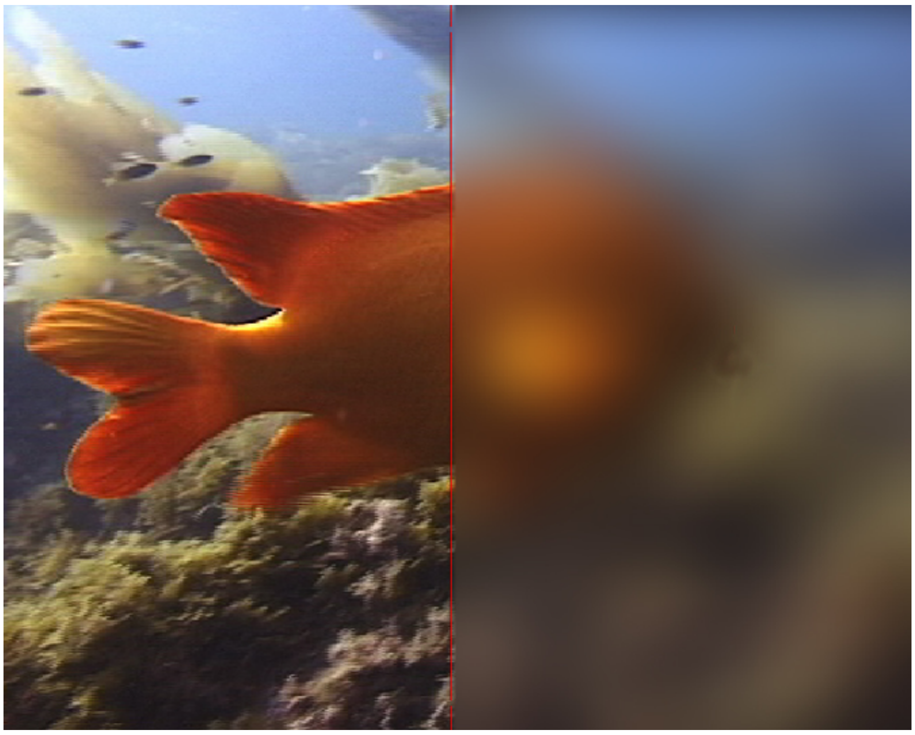
Figure 1. Source image (left) and after Gaussian blur filter applying (right).

1. Introduction

Gaussian blur is one of the most common image processing filters. Using it one can smooth image, simplify following computer-based image processing and improve the quality of objects recognition. But the mask of this filter is isotropic, so it limitates the area of filter usage. Logical extention of the Gaussian blur filter is bilateral filter, proposed by Tomasi [1].

Bilateral filtering problem could be easily parallelized - each pixel could be processed independently.

The new value of pixel intensity is calculated using the following formula:

$$h(a_0) = k^{-1} \sum_{i=0}^{n-1} f(a_i) \times g(a_i) \times r(a_i)$$

Where a_i characterizes intensity values in neigbour pixels, n – the number of pixels used in filter mask, k – normalizing constant to prevent intensity increase:

$$k = \sum_{i=0}^{n-1} g(a_i) \times r(a_i)$$

g(x,y) – coefficient depending on distance to a central pixel:

$$g(x,y) = e^{-\frac{x^2 - y^2}{\sigma_d^2}}$$

$r(a_i)$ – function calculating new intensity without normalizing coefficients.

Here σ_d is a parameter determining Gauss function amplitude,

$$r(a_i) = e^{\frac{(f(a_i)-f(a_0))^2}{\sigma_r^2}}$$

Where σ_r - rang-filter constant.

2. Task definition

Given the image of size M×N, implement and apply a CUDA version of 9-point bilateral filter and store the result to output image. Missing values for edge rows and columns are to be taken from nearest pixels. CUDA implementation must make use of texture memory.

3. Proposed method

The following method could be used to implement the bilateral filter:

1. Copy input data to device memory;
2. Bind input data to a texture link;
3. Extract each pixel together with its surrounding pixels via texture memory into 9-elements array;
4. Calculate the result pixel intensity using the formulas above;
5. Store the result into the array.

4. Implementation requirements

4.1. Input data

- Input grayscale image in BMP format, σ values;

4.2. Output data

- The time of image processing using GPU;
- The time of image processing using CPU;
- Resulting images in BMP format.

4.3. Implementation

The program is required to work on Linux machine. The resulting image could be exported in BMP format, using many open-source libraries, for instance, EasyBMP:

```
#include "EasyBMP.h"
...
BMP AnImage;
AnImage.SetSize(WIDTH, HEIGHT);
for (int i = 0; i < WIDTH; i++)
    for (int j = 0; j < HEIGHT; j++)
    {
        RGBApixel pixel;
        pixel.Red = pR[j * WIDTH + i];
        pixel.Green = pG[j * WIDTH + i];
        pixel.Blue = pB[j * WIDTH + i];
        pixel.Alpha = 0;
        AnImage.SetPixel(i, j, pixel);
    }
AnImage.WriteToFile(FILENAME);
```

CUDA implementation must use the texture memory.

5. Expected result

1. Getting familiar with CUDA applications development and texture memory.

References

[1] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images.