Salt-and-pepper noise filtering on GPU

Contents

1. Introduction **2**

2. Task definition **2**

3. Proposed method **2**

4. Implementation requirements **3**

4.1. Input data 3

4.2. Output data 3

4.3. Implementation 3

5. The expected result **3**

References **3**

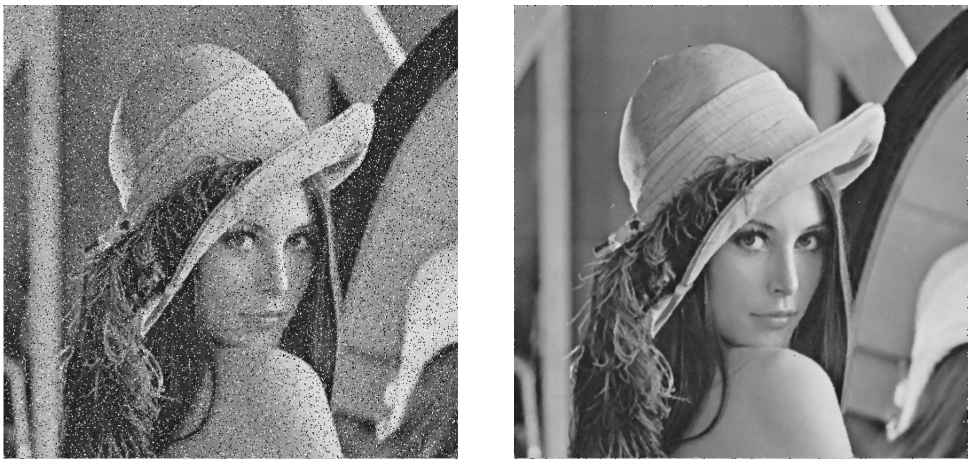

Figure 1. Images with “Salt and Pepper” noise (on the left) and filtered with median filter (on the right).

1. Introduction

One of the most common types of noise in digital photo and video is “Salt and pepper” noise of random white and black pixels. A number of filters could be used to remove this type of noise: median filters [2], morphological filters [1] and contraharmonic filters [4].

Median filter for “Salt and pepper” noise could be naturally parallelized: each pixel could be processed independently.

2. Task definition

Given the image of size M×N with “Salt and Pepper” noise, implement and apply a CUDA version of 9-point median 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 median 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. Sort array using any simple method, for instance the “Bubble” method;
5. Store the median element into the output array.

4. Implementation requirements

4.1. Input data

- The input grayscale image in BMP format;

4.2. Output data

- The time of image processing using GPU;
- The resulting image 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 [3]:

```
#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. The expected result

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

References

[1] Mathematical morphology – http://en.wikipedia.org/wiki/Mathematical_morphology.

[2] Median filter – http://en.wikipedia.org/wiki/Median_filter.

[3] Easybmp cross-platform windows bitmap library – http://easybmp.sourceforge.net/.

[4] Rafael C. Gonzalez and Richard E.Woods. *Digital image processing*. Prentice Hall, 2007.