KWANGWOON UNIVERSITY

디 지 털 영 상 처 리

HW #2

담당 교수님 : 이윤구 교수님

제 출 날 짜 : 2018. 10. 13.

학 과 : 컴퓨터소프트웨어학과

학 번 : 2014707040

이 름 : 유 진 혁

코드 설명 (C++ / Visual Studio 2017)

(1) HW #2 – 1

- Histogram Equalization

```
// Transfer function
unsigned char Transfer(unsigned char r)
{
    // Integrate the histogram from 0 to r.
    double sum = 0.0;
    for (int i = 0; i <= r; i++)
        sum += histogram[i];
    return (unsigned char)((PIXEL - 1)*sum);    // s = T(r)
}
```

<그림 1. Transfer 함수>

Histogram Equalization 에서 Transfer 함수를 구현한 것이다. Normalized histogram 값을 histogram 배열에 채운 뒤, 픽셀 값을 해당 함수의 인자로 주면 Histogram Equalization 된 픽셀 값이 반환된다.

```
// Histogram Equalization
string HistogramEqualize(string inputFilename, string outputFilename = "output.raw")
{
    ifstream fin;
    ofstream fout;
    unsigned char input[HEIGHT][WIDTH] = { 0 };
    unsigned char output[HEIGHT][WIDTH] = { 0 };

    // Read a image and produce a histogram.
    fin.open(inputFilename, ios::binary);
    for (int i = 0; i < HEIGHT; i++)
        for (int j = 0; j < WIDTH; j++)
        {
            input[i][j] = fin.get();
            histogram[input[i][j]]++;
        }
    fin.close();

    // Histogram normalize
    for (int i = 0; i < PIXEL; i++)
        histogram[i] /= (WIDTH * HEIGHT);

    // Write a image.
    fout.open(outputFilename, ios::binary);
    for (int i = 0; i < HEIGHT; i++)
        for (int j = 0; j < WIDTH; j++)
        {
            // Assign new pixel values.
            output[i][j] = Transfer(input[i][j]);
            fout << output[i][j];
        }
    fout.close();

    return outputFilename;
}
```

<그림 2. HistogramEqualize 함수>

입력 영상을 받아서 Histogram Equalization 을 한 뒤, 결과 영상을 저장하는 함수이다. 입력 영상 파일명을 인자로 받아 열고, 픽셀 값을 input 배열에 저장하면서 동시에 histogram 배열도 채운다. histogram 배열이 다 채워지면 각 값을 영상 크기로 나눠 Normalize 한다. 그리고 input 배열에 들어있는 픽셀 값을 Transfer 함수로 넘겨 새 픽셀 값을 받아 output 배열에 저장한다. output 배열의 값은 함수 두 번째 인자로 들어온 파일명으로 파일 출력된다.

(2) HW #2 – 2

- Spatial Filter (1) Average Filter

```
// Apply filter1 to the image.
void AverageFilter1()
{
    memset(output, 0, WIDTH * HEIGHT);  // for initializing the output array
    double filter1[3][3];
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            filter1[i][j] = 1.0 / 9.0;

    // Spatial filtering
    for (int i = 1; i < HEIGHT - 1; i++)
        for (int j = 1; j < WIDTH - 1; j++)
        {
            double sum = 0.0;
            for (int s = -1; s <= 1; s++)
                for (int t = -1; t <= 1; t++)
                    sum += (filter1[s + 1][t + 1] * input[i + s][j + t]);
            output[i][j] = (unsigned char)sum;
        }
}
```

<그림 3. AverageFilter1 함수>

input 배열에 Filter1 을 적용하고 그 결과를 output 배열에 저장하는 함수이다. 우선, 전역적으로 선언된 output 배열 값을 전부 0 으로 초기화 시키고 Filter1 배열도 선언 후 값을 채워준다. 그리고 output 배열을 기준으로 반복문을 돌리는데, 입력 영상 픽셀 값이 저장되어 있는 input 배열과 Filter1 의 값으로 Spatial Filtering 을 한다. Spatial Filtering 의 결과가 외곽을 제외한 output 배열에 채워진다.

```
// Apply filter2 to the image.
void AverageFilter2()
{
    memset(output, 0, WIDTH * HEIGHT);  // for initializing the output array
    double filter2[7][7];
    for (int i = 0; i < 7; i++)
        for (int j = 0; j < 7; j++)
            filter2[i][j] = 1.0 / 49.0;

    // Spatial filtering
    for (int i = 3; i < HEIGHT - 3; i++)
        for (int j = 3; j < WIDTH - 3; j++)
        {
            double sum = 0.0;
            for (int s = -3; s <= 3; s++)
                for (int t = -3; t <= 3; t++)
                    sum += (filter2[s + 3][t + 3] * input[i + s][j + t]);
            output[i][j] = (unsigned char)sum;
        }
}
```

<그림 4. AverageFilter2 함수>

input 배열에 Filter2 를 적용하고 그 결과를 output 배열에 저장하는 함수이다. 적용되는 필터의 값과 크기를 제외하고 AverageFilter1 과 같은 구조의 함수이다. 필터의 크기가 7x7 이므로 값이 채워지지 않는 output 배열의 외곽도 3 줄로 늘어난다.

- Spatial Filter (2) Smooth Filter

```
// Apply filter3 to the image.
void SmoothFilter()
{
    memset(output, 0, WIDTH * HEIGHT);  // for initializing the output array
    double filter3[3][3] = { {1.0 / 16.0, 2.0 / 16.0, 1.0 / 16.0},
                             {2.0 / 16.0, 4.0 / 16.0, 2.0 / 16.0},
                             {1.0 / 16.0, 2.0 / 16.0, 1.0 / 16.0} };

    // Spatial filtering
    for (int i = 1; i < HEIGHT - 1; i++)
        for (int j = 1; j < WIDTH - 1; j++)
        {
            double sum = 0.0;
            for (int s = -1; s <= 1; s++)
                for (int t = -1; t <= 1; t++)
                    sum += (filter3[s + 1][t + 1] * input[i + s][j + t]);
            output[i][j] = (unsigned char)sum;
        }
}
```

<그림 5. SmoothFilter 함수>

input 배열에 Filter3 를 적용하고 그 결과를 output 배열에 저장하는 함수이다. 이전의 함수들과 같은 구조이며, 다만 필터 안의 값들이 제각각 이므로 반복문을 쓰지 않고 필터 배열을 초기화하였다.

- Spatial Filter (3) Sharpening Filter

```
// Apply filter4 to the image result from filter1.
void SharpeningFilter()
{
    memset(output, 0, WIDTH * HEIGHT);  // for initializing the output array
    int filter4[3][3] = { {0, -1, 0},
                          {-1, 5, -1},
                          {0, -1, 0} };

    // Spatial filtering
    for (int i = 2; i < HEIGHT - 2; i++)
        for (int j = 2; j < WIDTH - 2; j++)
        {
            double sum = 0.0;
            for (int s = -1; s <= 1; s++)
                for (int t = -1; t <= 1; t++)
                    sum += (filter4[s + 1][t + 1] * input[i + s][j + t]);
            output[i][j] = (unsigned char)sum;
        }
}
```

<그림 6. SharpeningFilter 함수>

input 배열에 Filter4 를 적용하고 그 결과를 output 배열에 저장하는 함수이다. 필터의 값을 제외하고 SmoothFilter 함수와 구조가 같다. Filter1 의 결과 이미지에 Filter4 를 적용해야 하므로 외곽 픽셀을 제외하기 위해 output 배열의 (2,2) 지점부터 픽셀 값이 채워지도록 하였다.

- Spatial Filter (4) Median Filter

```
// Apply median filter to the images.
void MedianFilter(size_t maskSize)
{
    memset(output, 0, WIDTH * HEIGHT);      // for initializing the output array
    size_t size = maskSize * maskSize;      // total size of filter
    int startPoint = maskSize / 2;          // image filtering start point for input images
    vector<unsigned char> filterMask(size); // median filter

    // Spatial filtering
    for (int i = startPoint; i < HEIGHT - startPoint; i++)
        for (int j = startPoint; j < WIDTH - startPoint; j++)
        {
            for (int s = -startPoint; s <= startPoint; s++)
                for (int t = -startPoint; t <= startPoint; t++)
                    filterMask[(s + startPoint)*maskSize + (t + startPoint)] = input[i + s][j + t];
            sort(filterMask.begin(), filterMask.end()); // sorting
            output[i][j] = filterMask[size / 2];
        }
}
```

<그림 7. MedianFilter 함수>

인자로 들어온 값의 크기만큼 Filter Mask 를 만들고, 이를 이용해 Median Filter 를 input 배열에 적용한 뒤 output 배열에 결과를 저장하는 함수이다. Filter Mask 의 크기를 사용자가 변경 가능해야 하므로 filterMask 배열을 vector 로 선언하였다. 이 외의 과정은 이전까지의 함수와 유사하나 median 값을 구해야 하므로 중간에 배열을 정렬하는 부분이 들어간다. 그리고 Filter Mask 의 크기가 달라지므로 배열 인덱스 값들을 일반화하여 표현하였다.

- 공통 부분

```
// Read a image.
void ImageLoad(string inputFilename)
{
    ifstream fin;
    fin.open(inputFilename, ios::binary);
    for (int i = 0; i < HEIGHT; i++)
        for (int j = 0; j < WIDTH; j++)
        input[i][j] = fin.get();
    fin.close();
}

// Write a image.
void ImageSave(string outputFilename)
{
    ofstream fout;
    fout.open(outputFilename, ios::binary);
    for (int i = 0; i < HEIGHT; i++)
        for (int j = 0; j < WIDTH; j++)
            fout << output[i][j];
    fout.close();
}
```

<그림 8. ImageLoad 함수와 ImageSave 함수>

각각 인자로 들어온 파일명으로 파일을 열어 input 배열에 픽셀 값을 저장하는 함수와 output 배열에 저장된 픽셀을 출력하는 함수이다.

```
int main()
{
    int num;
    cout << "영상에 Spatial Filter를 적용합니다." << endl;
    cout << "(1) Average Filter" << endl;
    cout << "(2) Smooth Filter" << endl;
    cout << "(3) Sharpening Filter" << endl;
    cout << "(4) Median Filter" << endl;
    cout << ">> ";
```

<그림 9. main 함수에서의 메뉴 출력 부분>

전체적인 프로그램은 메뉴 방식으로 만들었다. 번호를 입력하면 해당하는 필터가 적용된 결과가 이미지 파일로 저장된다.

```
switch (num)
{
case 1:
    cout << "lena256.raw에 Filter1, Filter2를 적용합니다." << endl;
    cout << "..." << endl;

    ImageLoad("lena256.raw");

    AverageFilter1();
    ImageSave("average_filter_1.raw");

    AverageFilter2();
    ImageSave("average_filter_2.raw");

    cout << "결과 영상이 average_filter_1.raw, average_filter_2.raw로 저장되었습니다." << endl;
    break;
case 2:
    cout << "lena256.raw에 Filter3을 적용합니다." << endl;
    cout << "..." << endl;

    ImageLoad("lena256.raw");
    SmoothFilter();
    ImageSave("smooth_filter.raw");

    cout << "결과 영상이 smooth_filter.raw로 저장되었습니다." << endl;
    break;
```

<그림 10. main 함수에서의 switch문 일부>

사용자 입력에 따른 처리는 switch 문을 통해 구현하였고 과제의 요구 사항에 맞게 프로그램이 진행된다.

```
case 3:
    cout << "Filter1의 결과에 Filter3을 적용합니다." << endl;
    cout << "..." << endl;

    ImageLoad("lena256.raw");
    AverageFilter1();
    // The filter1 result image is replaced with the input image.
    for (int i = 0; i < HEIGHT; i++)
        for (int j = 0; j < WIDTH; j++)
            input[i][j] = output[i][j];

    SharpeningFilter();
    ImageSave("sharpening_filter.raw");

    cout << "결과 영상이 sharpening_filter.raw로 저장되었습니다." << endl;
    break;
```

<그림 11. main 함수에서의 switch문 일부>

Sharpening Filter 의 경우, Filter1 의 결과에 Filter4 를 적용해야 하므로, AverageFilter1 함수를 먼저 호출시킨 뒤 그 결과를 다시 input 배열에 넣는 과정이 포함되어 있다.

```
case 4:
    size_t size;

    cout << "Filter Mask 크기 (n X n, n은 홀수)" << endl;
    cout << "n = ";

    cin >> size;
    cout << endl;

    // only odd number
    if (size % 2 == 0)
    {
        cout << "입력 오류" << endl;
        break;
    }
    else
    {
        cout << "노이즈가 포함된 lena256_n5.raw, lena256_n10.raw, lena256_n25.raw에 Median Filter를 적용합니다." << endl;
        cout << "..." << endl;

        ImageLoad("lena256_n5.raw");
        MedianFilter(size);
        ImageSave("median_filter_n5.raw");

        ImageLoad("lena256_n10.raw");
        MedianFilter(size);
        ImageSave("median_filter_n10.raw");

        ImageLoad("lena256_n25.raw");
        MedianFilter(size);
        ImageSave("median_filter_n25.raw");

        cout << "결과 영상이 median_filter_n5.raw, median_filter_n10.raw, median_filter_n25.raw로 저장되었습니다." << endl;
        break;
    }
```

<그림 12. main 함수에서의 switch문 일부>

Median Filter 의 경우, 사용자가 Filter Mask 의 크기를 지정해야 하므로 사용자 입력을 한 번 더 받는다. 이 때, Median Filter 를 적용할 수 없는 짝수가 입력되면 오류를 표시하며 프로그램이 종료되게 하였다.

입력 영상과 결과 영상

(1) HW #2 – 1

- Histogram Equalization

입력 영상 (input.raw)

결과 영상

(2) HW #2 – 2

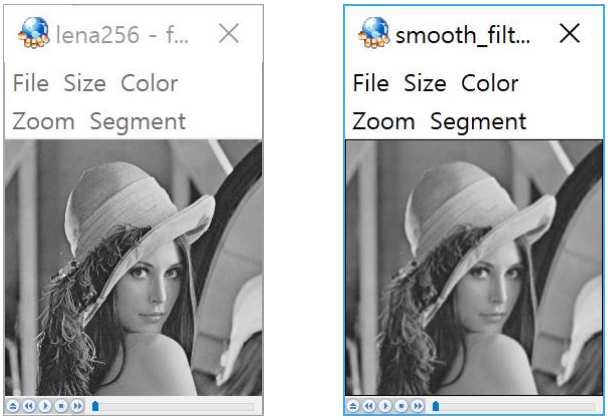
- Spatial Filter (1) Average Filter

입력 영상 (lena256.raw)

결과 영상 (Filter1 적용)

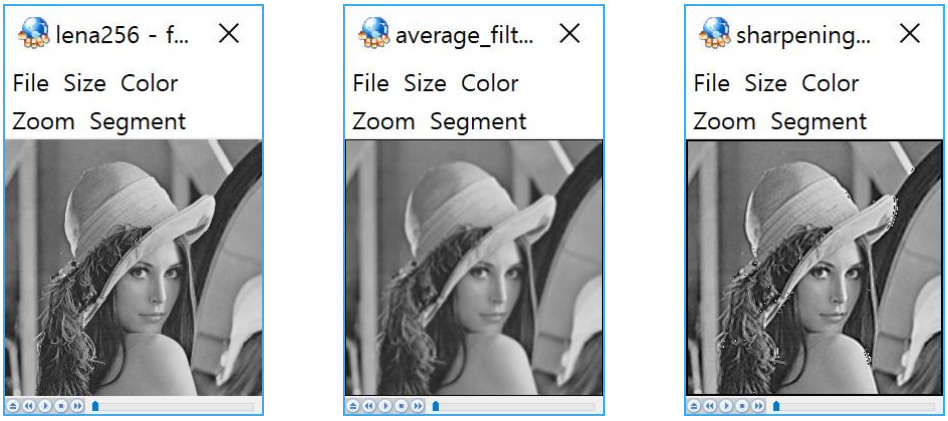
결과 영상 (Filter2 적용)

- Spatial Filter (2) Smooth Filter



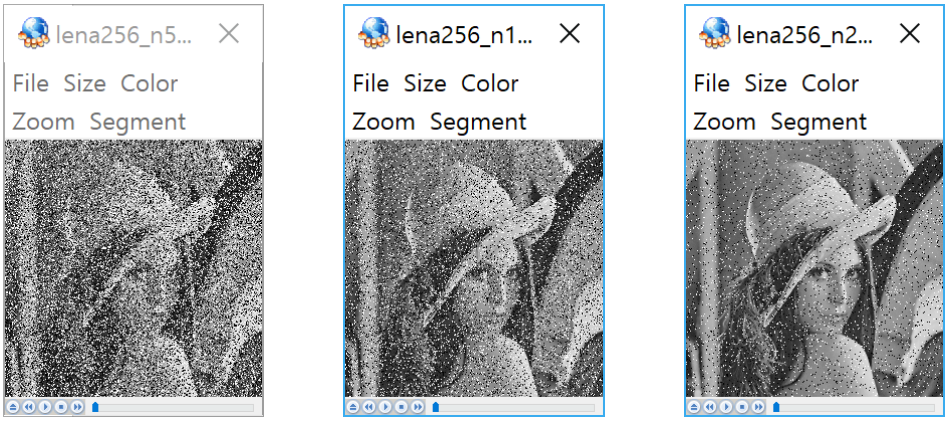
입력 영상 (lena.256raw) 결과 영상 (Filter 3 적용)

- Spatial Filter (3) Sharpening Filter



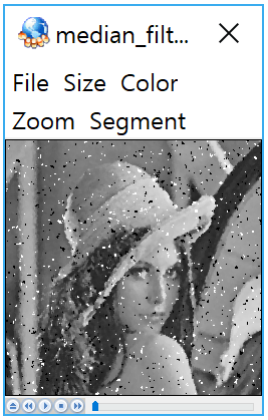
입력 영상 (lena256.raw) 입력 영상 (Filter1의 결과) 결과 영상 (Filter 4 적용)

- Spatial Filter (4) Median Filter



입력 영상 n5 입력 영상 n10 입력 영상 n25

Filter Mask 크기 3x3



결과 영상 n5

결과 영상 n10

결과 영상 n25

Filter Mask 크기 7x7

결과 영상 n5

결과 영상 n10

결과 영상 n25

고찰

처음에는 어떻게 접근해야 할지 막막했지만 강의 자료의 내용을 충실히 따라가니 구현할 수 있었다. Histogram Equalization 부분은 먼저 입력 영상을 분석하는 일부터 시작했다. 입력 영상을 읽어들이고 픽셀 값들을 분석해보니 픽셀 값의 범위가 0부터 63까지의 값만 갖는 것을 확인할 수 있었다. 이를 토대로 Normalized Histogram 배열을 만들고 강의 자료의 내용처럼 Transfer 함수를 구현하니 Histogram Equalization된 픽셀 값이 반환되었다. Histogram Equalization을 진행한 뒤에도 히스토그램 전체 내부 면적이 같은가를 확인해보니 면적 값이 이상없이 1이었다. 입력 영상은 픽셀 범위가 넓어졌으므로 대조가 큰 영상이 되었다.

Spatial Filtering도 마찬가지로 강의 자료에 나온 Spatial Correlation 수식을 그대로 적용하여 구현할 수 있었다. 다만, 구현을 하면서 몇 가지 궁금한 점이 생겼는데, 첫 번째는 외곽 픽셀에 대한 처리 방법이다. 수업 시간에 다루지 않았던 부분이라 여러 웹 문서를 찾아보았는데, 다양한 방법들이 있었다. 가장 간단한 방법은 그냥 결과 영상에서 외곽 픽셀의 값을 0으로 두는 것이었고, 또 다른 방법은 이미지 영역을 벗어나는 부분의 픽셀 값을 0으로 가정하고 처리하는 것이었다. 그 밖에도 이미지 영역을 벗어나는 부분의 픽셀 값을 현재 보고 있는 위치의 픽셀 값으로 채우거나, Interpolation을 하는 방법 등 다양했다. 어느 방법이 가장 좋은 방법이고 많이 사용되는지는 더 공부를 해야 할 것 같다. 두 번째로 궁금한 점은 Sharpening Filter 적용 시 생기는 노이즈의 원인이다. Sharpening Filter를 적용하니 결과 영상 곳곳에 튀는 픽셀들이 발생하였다. 다른 Filter들과 다르게 왜 이런 노이즈가 생겼는지를 생각해보았는데, 아마 필터에 음수 값이 있어서 발생한 것으로 추측해본다. 픽셀 값은 unsigned char형으로 음수 값을 가질 수 없는데, Sharpening Filter를 적용하다가 우연히 값들이 잘 맞아 떨어져 결과 값이 음수가 된 것 같다. 이를 unsigned char형으로 캐스팅하면서 노이즈가 발생한 것으로 보인다.

그리고 과제를 하면서 어려웠던 부분은 어떤 필터를 적용했을 때 어떤 결과 영상이 나올지 잘 예측이 되지 않았다는 점이다. 이를 위해 앞으로 영상에 여러 Spatial Filter를 적용해보면서 생각해보고 익히며 공부를 계속해야겠다.