KWANGWOON UNIVERSITY

디 지 털 영 상 처 리

Homework #1

담당 교수님 : 이윤구 교수님

제 출 날 짜 : 2018. 10. 07.

학 과 : 컴퓨터소프트웨어학과

학 번 : 2014707040

이 름 : 유 진 혁

코드 설명 (C++ / Visual Studio 2017)

0) 공통 부분

```cpp
// Linear Interpolate Operation
unsigned char LinearInterpolate(unsigned char p1, unsigned char p2, double a)
{
    return (unsigned char)((1 - a) * p1 + a * p2);
}
```

<그림 1. LinearInterpolate 함수>

Interpolation 방법 중 Linear Interpolation을 수행하는 함수이다. 두 개의 입력 픽셀 값과 보간 픽셀과의 거리 값을 인자로 주면 보간된 픽셀 값을 계산하여 반환한다.

```cpp
// Bilinear Interpolate Operation (Interpolates the value of the coordinates x and y of the image.)
unsigned char BilinearInterpolate(vector< vector<char> > &image, double x, double y)
{
    unsigned int pixelX = (unsigned int)x;
    unsigned int pixelY = (unsigned int)y;
    if (pixelX >= 0 && pixelX < image[0].size() && pixelY >= 0 && pixelY < image.size())
    {
        unsigned char p1, p2, p3, p4, value;
        // For not the right-end & the bottom-end pixel
        if ((pixelX < image[0].size() - 1) && (pixelY < image.size() - 1))
        {
            p1 = image[pixelY][pixelX];
            p2 = image[pixelY][pixelX + 1];
            p3 = image[pixelY + 1][pixelX];
            p4 = image[pixelY + 1][pixelX + 1];
            value = LinearInterpolate(LinearInterpolate(p1, p2, x - pixelX), LinearInterpolate(p3, p4, x - pixelX), y - pixelY);
        }
        // For the right-end pixel
        else if ((pixelX == image[0].size() - 1) && (pixelY < image.size() - 1))
        {
            p1 = image[pixelY][pixelX];
            p3 = image[pixelY + 1][pixelX];
            value = LinearInterpolate(p1, p3, y - pixelY);
        }
        // For the bottom-end pixel
        else if ((pixelX < image[0].size() - 1) && (pixelY == image.size() - 1))
        {
            p1 = image[pixelY][pixelX];
            p2 = image[pixelY][pixelX + 1];
            value = LinearInterpolate(p1, p2, x - pixelX);
        }
        // For the right-end & the bottom-end pixel
        else if ((pixelX == image[0].size() - 1) && (pixelY == image.size() - 1))
        {
            value = image[pixelY][pixelX];
        }

        return value;
    }

    return 0;
}
```

<그림 2. BilinearInterpolate 함수>

Bilinear Interpolation을 할 좌표와 그 이미지를 입력 인자로 주면 보간된 픽셀 값을 반환하는 함수이다. 인자로 들어온 좌표와 인접한 픽셀 값을 참조하여 Linear Interpolation을 재귀적으로 수행한다. 외곽의 픽셀은 상하, 혹은 좌우 픽셀 값만 참조하여 Interpolation하고, 오른쪽 맨 아래 픽셀의 경우, 입력 좌표 픽셀 값을 그대로 결과에 반환한다.

1) HW #1 -1 Scaling

```
// Enlarge a image
string Scale(string inputFilename, int imgWidth, int imgHeight, double x, string outputFilename = "output.raw")
{
    ifstream fin;
    ofstream fout;
    vector< vector<char> > input(imgHeight, vector<char>(imgWidth));
    int newWidth = (int)(x * imgWidth);
    int newHeight = (int)(x * imgHeight);
    vector< vector<char> > output(newHeight, vector<char>(newWidth));

    // Read a image
    fin.open(inputFilename, ios::binary);
    for (int i = 0; i < imgHeight; i++)
        for (int j = 0; j < imgWidth; j++)
            fin.get(input[i][j]);
    fin.close();

    // Write a image
    fout.open(outputFilename, ios::binary);
    for (int i = 0; i < newHeight; i++)
        for (int j = 0; j < newWidth; j++)
        {
            // Calculate the coordinates when the enlarged image is set to the input image size.
            double interpolatedX = j * (imgWidth - 1) / (newWidth - 1);
            double interpolatedY = i * (imgHeight - 1) / (newHeight - 1);
            output[i][j] = BilinearInterpolate(input, interpolatedX, interpolatedY);
            fout << output[i][j];
        }
    fout.close();

    return outputFilename;
}
```

<그림 3. Scale 함수>

확대시킬 입력 영상의 파일명, 영상의 크기, 확대 배수, 출력 영상의 파일명을 함수 인자로 받는다. 출력 파일명을 지정하지 않을 경우 output.raw로 저장이 된다. 입력 영상과 출력 영상을 담을 배열이 vector로 선언되어 있고, 출력 영상의 vector는 입력 영상의 x배한 크기로 만들어진다. 파일 스트림을 열어 lena256.raw의 바이너리 정보를 input 배열에 담는다. 그리고 확대한 영상을 입력 영상 크기에 맞췄을 때의 각 좌표를 계산하고, 그 값을 BilinearInterpolation 함수에 넣어 반환값으로 output 배열을 채운다. 이어서 output 배열에 들어 있는 값을 파일 스트림으로 출력한다. Scale 함수는 출력 영상의 파일명을 문자열로 반환한다.

```
int main()
{
    double x;
    cout << "lena256.raw를 확대 시킵니다. 몇 배 확대 시킬까요?" << endl;
    cout << "(436x436 : 1.703125, 512x512 : 2)" << endl;
    cout << ">> ";
    cin >> x;
    cout << "변환 영상이 " << Scale("lena256.raw", 256, 256, x) << "(으)로 저장되었습니다." << endl;
    return 0;
}
```

<그림 4. main 함수>

표준 입력으로 확대 배수를 입력 받고 내용을 출력한다.

2) HW #1 -2 Rotation

```
// Rotate a image
string Rotate(string inputFilename, int imgWidth, int imgHeight, double x, string outputFilename = "output.raw")
{
    ifstream fin;
    ofstream fout;
    vector< vector<char> > input(imgHeight, vector<char>(imgWidth));
    vector< vector<char> > output(imgHeight, vector<char>(imgWidth));

    // Read a image
    fin.open(inputFilename, ios::binary);
    for (int i = 0; i < imgHeight; i++)
        for (int j = 0; j < imgWidth; j++)
            fin.get(input[i][j]);
    fin.close();

    // Degree to Radian
    double rad = x * M_PI / 180;

    // Write a image
    fout.open(outputFilename, ios::binary);
    for (int i = 0; i < imgHeight; i++)
        for (int j = 0; j < imgWidth; j++)
        {
            // Rotation transformation
            double rotatedX = (cos(rad) * (j - (imgWidth - 1) / 2) + sin(rad) * (i - (imgHeight - 1) / 2)) + (imgWidth - 1) / 2;
            double rotatedY = (-sin(rad) * (j - (imgWidth - 1) / 2) + cos(rad) * (i - (imgHeight - 1) / 2)) + (imgHeight - 1) / 2;
            output[i][j] = BilinearInterpolate(input, rotatedX, rotatedY);
            fout << output[i][j];
        }
    fout.close();

    return outputFilename;
}
```

<그림 5. Rotate 함수>

회전시킬 입력 영상의 파일명, 영상의 크기, 회전 각도, 출력 영상의 파일명을 함수 인자로 받는다. 출력 파일명을 지정하지 않을 경우 output.raw로 저장이 된다. 입력 영상과 출력 영상을 담을 배열이 vector로 선언되어 있다. 파일 스트림을 열어 lena256.raw의 바이너리 정보를 input 배열에 담는다. 입력으로 받은 회전 각도를 60분법에서 호도법으로 변환한다. 그리고 회전 변환을 한 좌표를 계산하고, 그 값을 Bilinear Interpolation하여 output 배열을 채운다. 이어서 output 배열에 들어 있는 값을 파일 스트림으로 출력한다. Rotate 함수는 출력 영상의 파일명을 문자열로 반환한다.

```
int main()
{
    double x;
    cout << "lena256.raw를 회전 시킵니다. 몇 도 회전 시킬까요?" << endl;
    cout << "(30, 45, 60)" << endl;
    cout << ">> ";
    cin >> x;
    cout << "변환 영상이 " << Rotate("lena256.raw", 256, 256, x) << "(으)로 저장되었습니다." << endl;
    return 0;
}
```

<그림 6. main 함수>

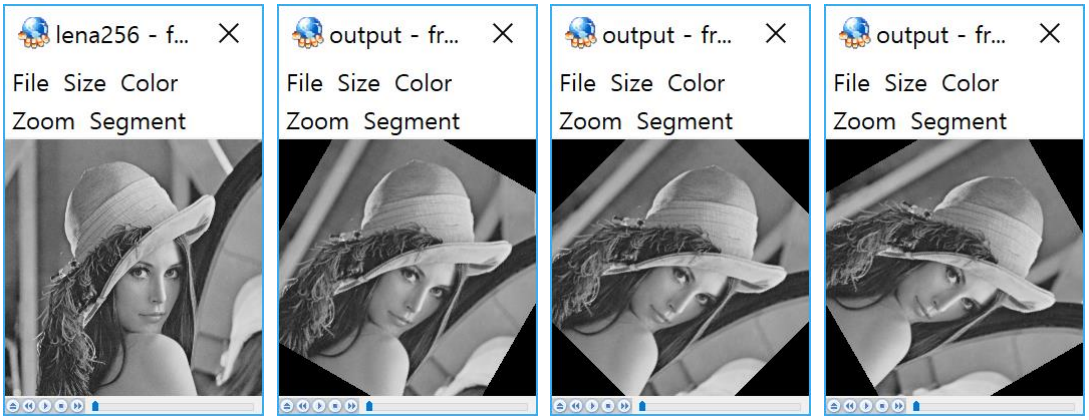
표준 입력으로 회전 각도를 입력 받고 내용을 출력한다.

결과

1) Scaling

원본(256x256) 1.703125배(436x436) 2배(512x512)

2) Rotation



0도 30도 45도 60도

고찰

Scaling부터 구현을 시작했다. 일반화와 규칙성을 찾아내기 위해 작은 크기의 영상을 기준으로 좌표 변화를 직접 계산해보았다. 영상의 크기가 커지면 각 픽셀들은 어디로 이동하는지, 기존 픽셀과의 거리는 어떻게 변화하는지 찾아내고 일반화된 식을 작성하였다. 강의 자료의 내용을 토대로 Bilinear interpolation을 수행하는 함수를 작성하고 인덱스 범위가 넘어가는 외곽 픽셀은 범위가 넘지 않는 인접한 픽셀을 이용하여 Linear interpolation을 해주었다. 그 결과, 영상 확대 시 빈 픽셀과 어색한 부분이 없는 부드러운 영상이 만들어졌다.

Rotation 구현에는 많은 시행 착오를 겪었다. Scaling과는 달리 입력 영상을 기준으로 입력 영상 좌표에 회전 변환을 주어 출력 영상을 만드니 곳곳에 빈 픽셀들이 생겼다. 입력 영상 좌표에 회전 변환을 주어 인덱스로 사용하면 자료형이 정수로 바뀌어서 출력 영상에 채워지지 않은 부분이 생기는 것이다. 그래서 Scaling처럼 출력 영상을 기준으로 두고 반대로 계산을 하여 출력 영상에 빈 픽셀이 생기지 않도록 구현을 하였다. 그 결과, 영상에 빈 공간은 없었으나, Interpolation을 하지 않아 부드러운 영상이 출력되지 않았다. Scaling 구현에서 사용했던 방법 그대로 Bilinear interpolation을 적용해줬는데 이번에 곳곳에 깨진 픽셀들이 생겼다. 이 문제를 해결하는데 꽤 오랜 시간이 걸렸는데, 원인은 픽셀 값으로 unsigned char형이 아닌 char형 데이터를 받았기 때문이다. Scaling에선 발견되지 않았던 문제로, 회전 영상을 interpolation하면서 픽셀에 음수 값이 들어가 픽셀이 깨진 것으로 보인다. 이를 해결하니 영상을 회전하여도 깨진 픽셀과 빈 픽셀이 없는 부드러운 영상이 출력되었다. Scaling과 Rotation에서 공통적으로 쓰인 부분을 별도의 함수로 빼서 코드를 위와 같이 정리했다.