디지털 영상처리 hw2

2011903054 행정학과

문석현

<과제 구현 완료 여부 작성>

1번: 작성

2번 1) 작성

2번 2) 실패(75%완료)

2번 3) 실패 (75% 완료)

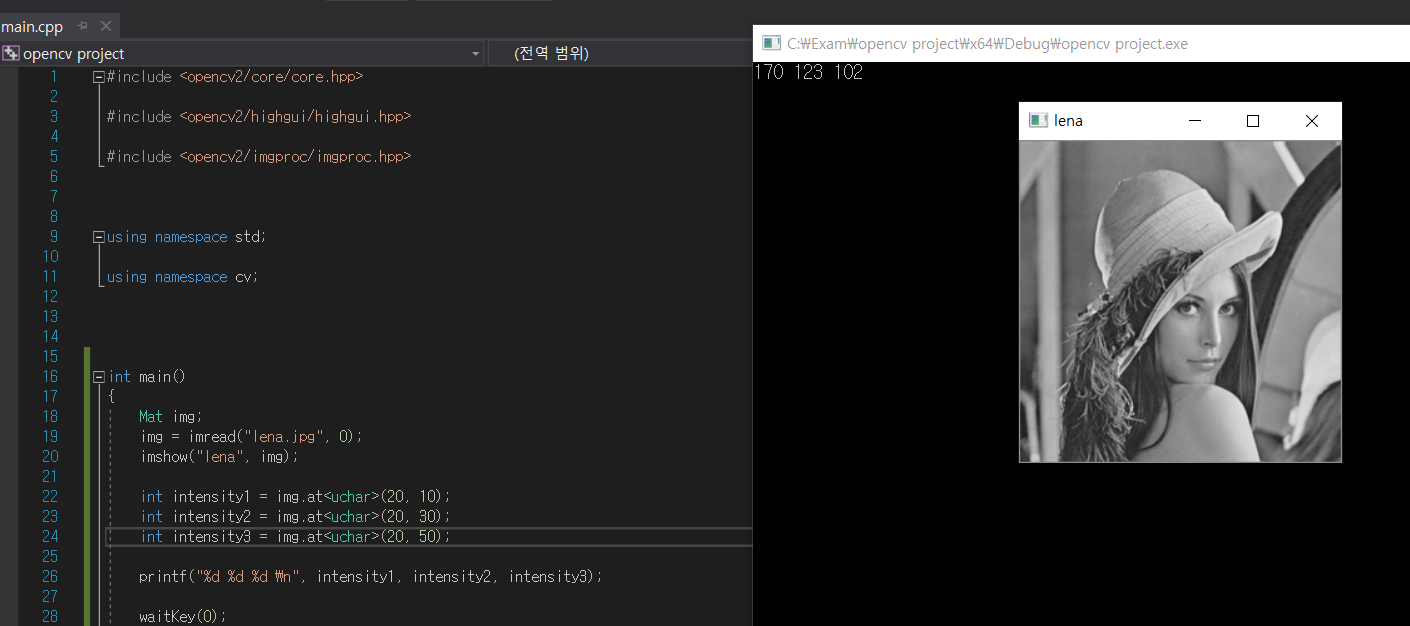
3번 1) 성공

3번 2) (1) 실패(50% 완료)

(2) 실패(50% 완료)

1. RGB 영상 포맷 확인

(1) 그레이 영상에서 이미지픽셀에 접근했을 때 특정 픽셀에서 접근 값



#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/imgproc/imgproc.hpp>

// 이하 Mat 그리고 imread를 사용하기 위한 라이브러리를 불러옴

using namespace std;

using namespace cv;

int main()

{

Mat img;

img = imread("lena.jpg", 0); // Mat 함수를 선언한 후에 imread를 통해 lena 이미지를 불러옴

imshow("lena", img);

int intensity1 = img.at<uchar>(20, 10);

int intensity2 = img.at<uchar>(20, 30);

int intensity3 = img.at<uchar>(20, 50); // (20, 10) (20,30) (20,50)의 픽셀값들을 intensity 함수에 저장

printf("%d %d %d \n", intensity1, intensity2, intensity3);

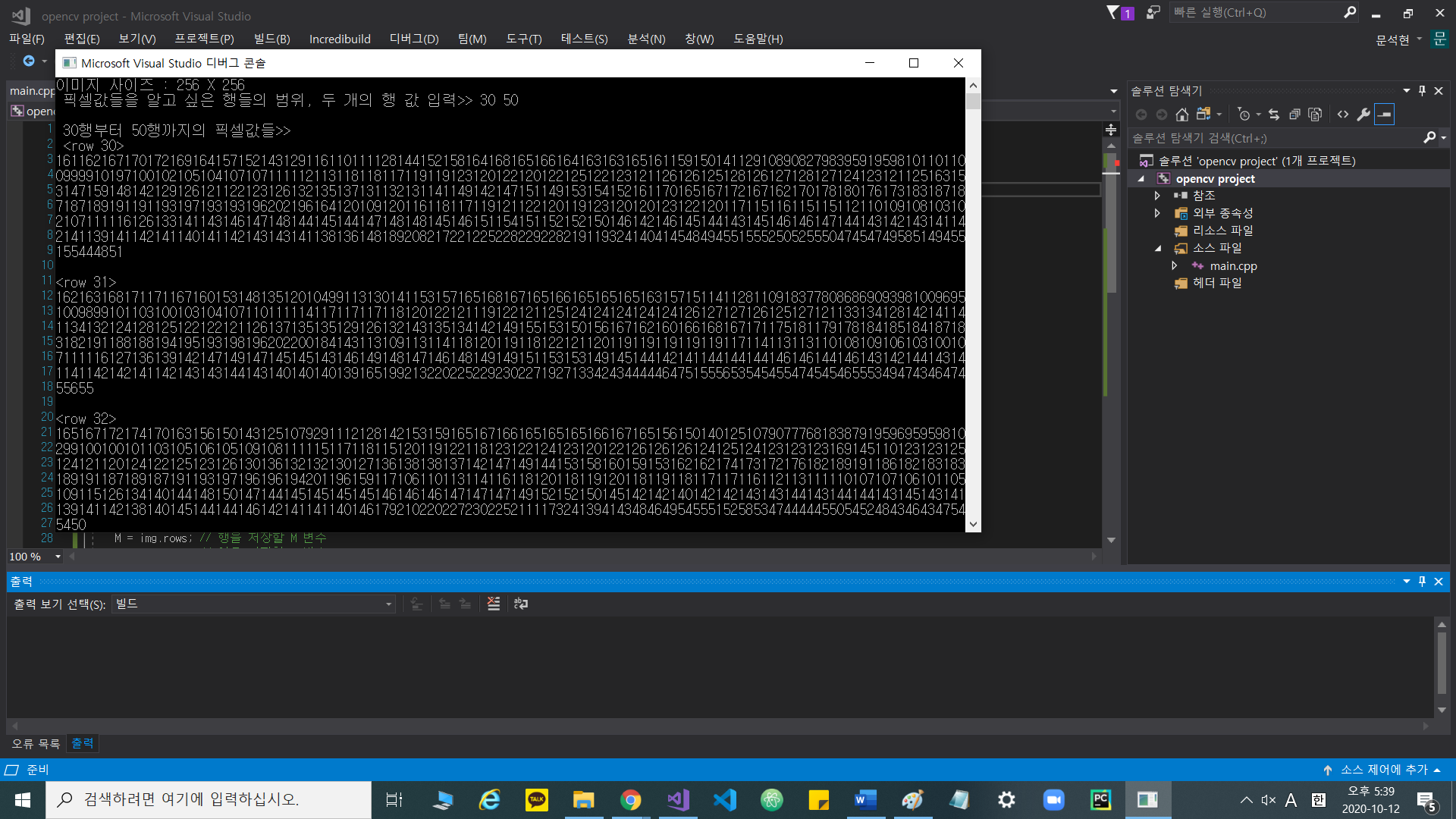
// printf 함수로 출력

waitKey(0);

return 0;

}

(2) 그레이 이미지를 행 단위로 접근해보는 방법



#define \_CRT\_SECURE\_NO\_WARNINGS

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/imgproc/imgproc.hpp>

using namespace std;

using namespace cv;

int main()

{

Mat img;

img = imread("lena.jpg", 0);

imshow("lena", img);

waitKey(0);

int M, N;

M = img.rows; // 행을 저장할 M 변수

N = img.cols; // 열을 저장할 N 변수

printf("이미지 사이즈 : %d X %d \n ", M, N);

int row1, row2; // row1부터 row2까지의 데이터를 출력하기 위한 변수 선언

printf("픽셀값들을 알고 싶은 행들의 범위, 두 개의 행 값 입력>> ");

scanf("%d %d", &row1, &row2);

printf("\n %d행부터 %d행까지의 픽셀값들>> \n ", row1, row2);

for (int i = row1; i <= row2; i++) // 행들의 경우 row1은 시작점이며 row2는 끝나는 부분이니 크거나 같다로 표현

{

printf("<row %d> \n", i);

for (int j = 0; j < N; j++)

{

printf("%d", img.at<uchar>(i, j));

}//행부터 행까지의 데이터를 for문을 통해 출력하기 위해 이중포문을 사용

printf("\n\n");

}

return 0;

}

[설명]

위의 방식을 통해 RGB 데이터의 경우 각각 행렬마다 이미지의 데이터 값이 숫자로 저장되어 있음을 알 수 있었습니다

2. YUV color space

(1) YUV color space에 대해 간략히 조사

출처: <https://ko.wikipedia.org/wiki/%EC%83%89_%EA%B3%B5%EA%B0%84>

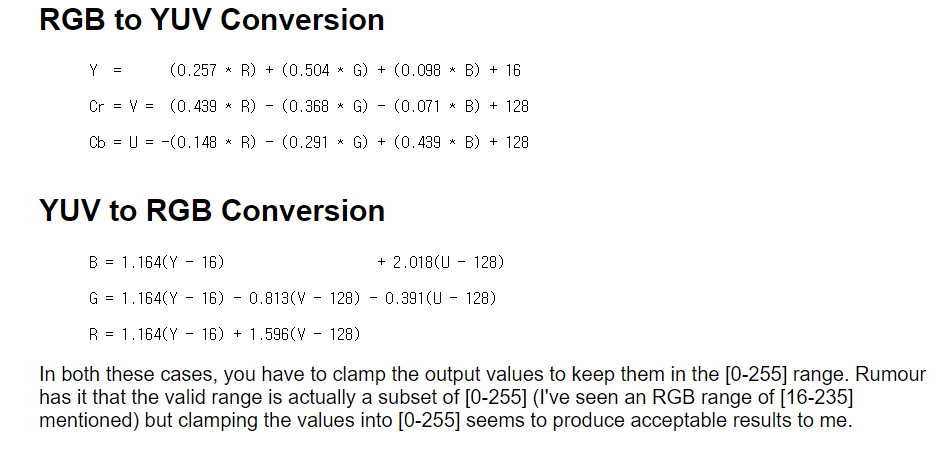
색공간(color space)이란 색상,명도,채도로 구성된 3차원의 공간의 각각의 축으로 형성되는 공간으로 빨강,노랑,보라,초록 등으로 구분되는 색상으로 색상각은 0도에서 ~360도의 범위를 가지며 시계방향으로 변화한다.

다음으로 모든 색들의 밝고 어두움을 나타내는 명도는 남극과 북극을 연결하는 축으로 남극을 검은색 , 북극을 흰색으로 하며 그 사이에는 회색으로 배열된다

다음은 채도로서 모든 색들의 깨끗한 정도를 나타내며 색 공간의 명도축을 0으로 하고 적도에 가까이 갈수록 커지게 된다.

(2) RGB를 YUV로 변환하는 코드 작성(실제 matrix연산을 통해 구할 것)

1) rgb to yuv 변환 공식



2) rgb to yuv

가)헤더파일

#pragma once

#pragma warning(disable:4996)

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#include<string.h>

#define WIDTH 352

#define HEIGHT 288

#define Clamp(x)(x<0?0:(x>255?255:x))

typedef unsigned char uchar;

int ReadFrame(FILE \*fin, uchar \*in, int width, int height);

int WriteFrame(FILE \*fout, uchar \*in, int width, int height);

void RGB2YUV444(uchar \*rgb, uchar \*yuv, int width, int height);

나) 함수 구현

#include "head.h"

using namespace std;

int ReadFrame(FILE \*fin, uchar \*in, int width, int height)

{

return fread(in, sizeof(uchar), width\*height, fin); /\* 스트림에서 바이너리 데이터를 읽어 버퍼에 저장하는

표준 라이브러리, 읽기 성공시 읽은 크기 반환\*/

}

void WriteFrame(FILE \*fout, uchar \*in, int width, int height)

{

fwrite(in, sizeof(uchar), width\*height, fout);

}

void RGB2YUV444(uchar \*rgb, uchar \*yuv, int width, int height)

{

uchar \*R, \*G, \*B; // RGB를 가리키는 포인터 변수 선언

R = rgb;

G = rgb + width \* height;

B = rgb + width \* height \* 2;

for (int j = 0; j < height; j++) // 이중 for문을 이용해 RGB를 YUV로 변환시키는 matrix구현

{

for (int i = 0; i < width; i++)

{

yuv[j\*width + i] = (uchar)((66 \* R[j\*width + i]) + 129 \* G[j\*width + i] + 25 \* B[j\*width + i] + 128 >> 8) + 16;

yuv[(j + height)\*width + i] = (uchar)((-38 \* R[j\*width + i] - 74 \* G[j\*width + i] + 112 \* B[j\*width + i] + 112 \* B[j\*width + i] + 128) >> 8) + 128;

yuv[(j + height \* 2)\*width + i] = (uchar)((112 \* R[j\*width + i] - 94 \* G[j\*width + i] - 18 \* B[j\*width + i] + 128) >> 8) + 128;

}

}

}

void main()

{

FILE \*fin = fopen("lena.jpg", "rb");

FILE \*fyuv = fopen("[YUV]lena.jpg", "wb");

FILE \*frgb = fopen("[RGB]lena.jpg", "wb");

uchar \*in, \*out;

int num = 1;

in = (uchar \*)malloc(sizeof(uchar)\*WIDTH\*HEIGHT \* 3);

out = (uchar \*)malloc(sizeof(uchar)\*WIDTH\*HEIGHT \* 3);

while (num = ReadFrame(fin, in, WIDTH, HEIGHT \* 3))

{

RGB2YUV444(in, out, WIDTH, HEIGHT \* 3);

WriteFrame(fyuv, out, WIDTH, HEIGHT \* 3);

}

free(in);

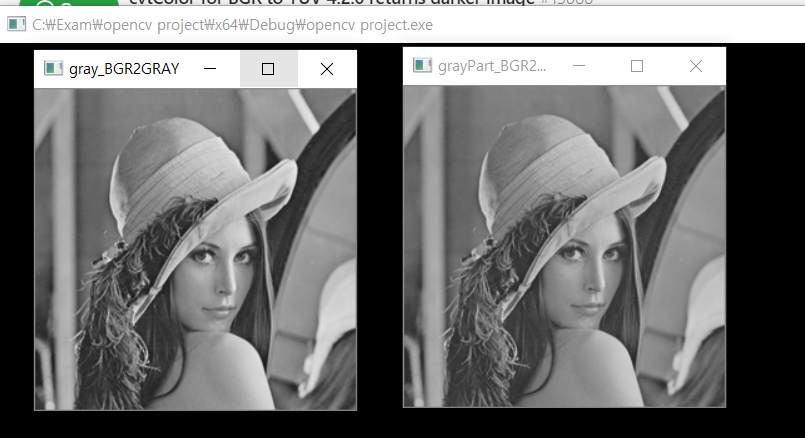
free(out);

fcloseall();

}

3) cvtcolor를 이용하여 YUV로 변환한 결과와 2)의 결과를 비교

가)cvtcolor를 이용한 변환



#include <math.h>

#include <iostream>

#include <opencv2/opencv.hpp>

#include <opencv2/highgui/highgui.hpp>

#include<opencv2/imgproc/imgproc.hpp>

using namespace std;

using namespace cv;

void main()

{

cv::Mat inputImage = cv::imread("lena.jpg");

cv::Mat gray\_BGR2GRAY;

cv::cvtColor(inputImage, gray\_BGR2GRAY, cv::COLOR\_BGR2GRAY);

imshow("gray\_BGR2GRAY", gray\_BGR2GRAY);

cv::Mat mat\_BGR2YUV\_I420;

cv::cvtColor(inputImage, mat\_BGR2YUV\_I420, cv::COLOR\_BGR2YUV\_I420);

cv::Mat grayPart\_BGR2YUV\_I420(mat\_BGR2YUV\_I420, cv::Rect(0, 0, inputImage.cols, inputImage.rows));

imshow("grayPart\_BGR2YUV\_I420", grayPart\_BGR2YUV\_I420);

cv::waitKey(0);

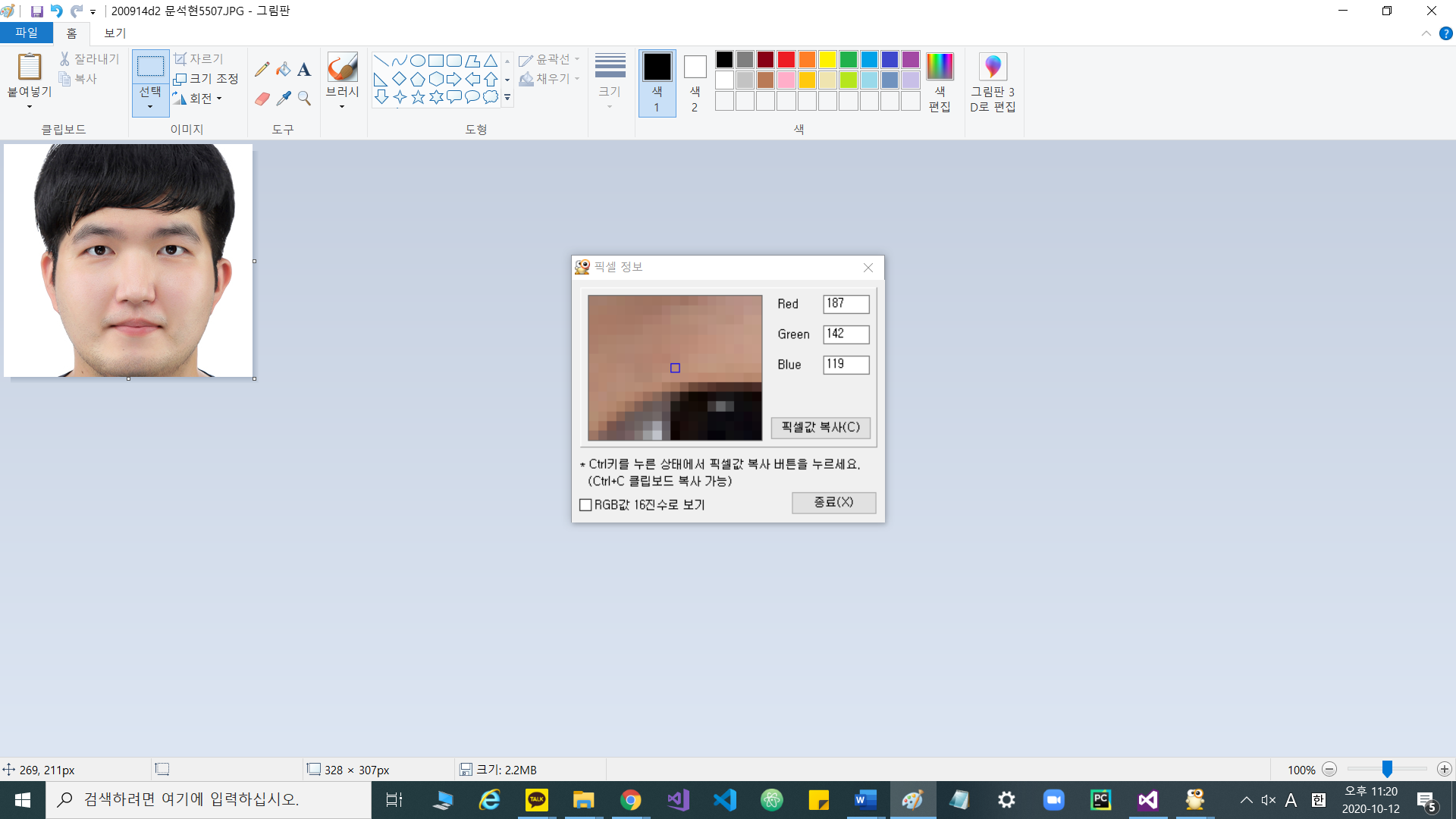
return ;

}

나) 2번을 이용한 변환 결과

3. face slicing

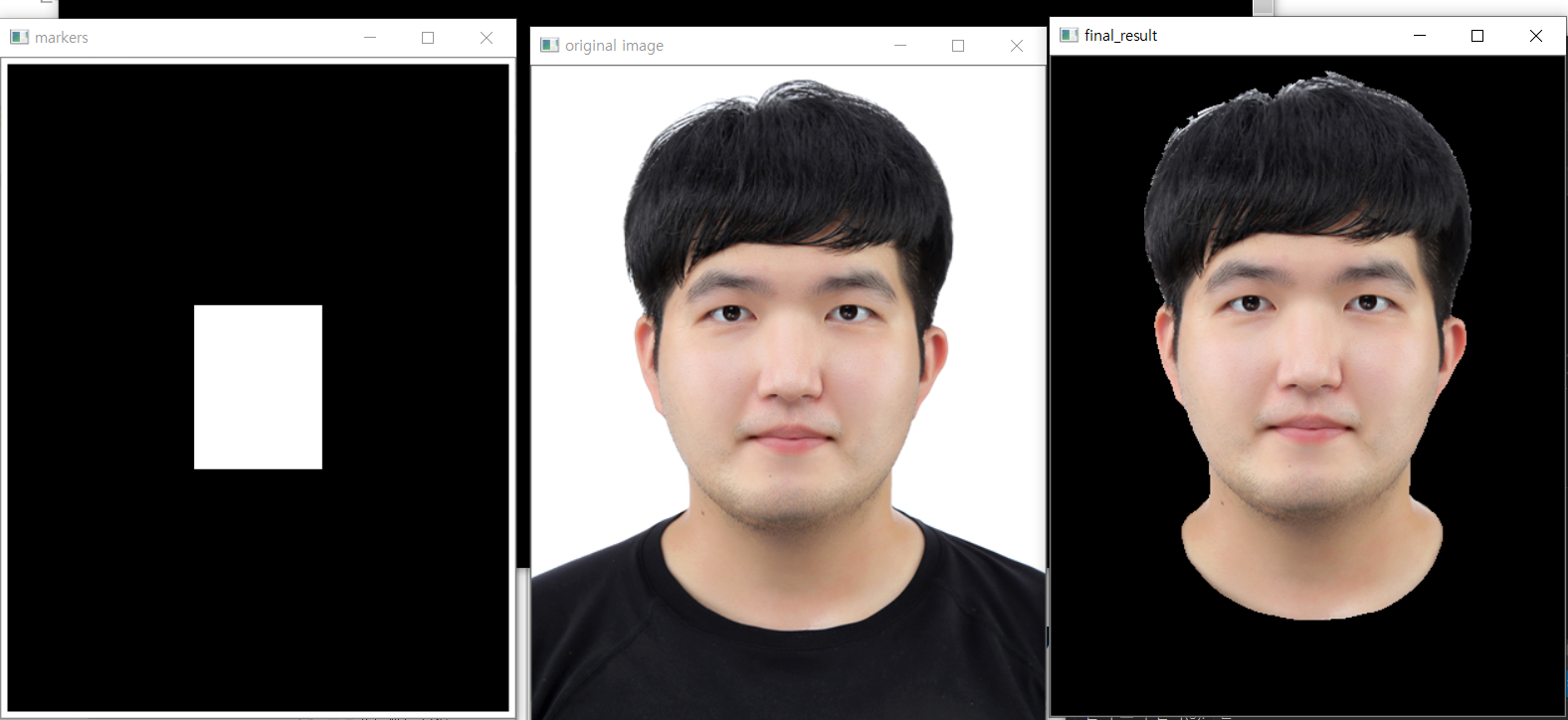
1) 얼굴영역에 대한 색정보 확인



2) 영상의 pixel 값을 접근하여 얼굴색 여부를 판단

원리

watershed알고리즘 이란 이미지를 grayscale로 전환하여 pixel 값(0~255)의 높고 낮음을 구분하여 이것을 통해 얼굴의 pixel값 그리고 얼굴외의 pixel값을 구분한 후 경계선을 이미지의 구분지점으로 파악하여 이미지 분할을 하게 되는 방식을 의미한다.



#include "opencv2/opencv.hpp"

#include "opencv2/highgui/highgui.hpp"

#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <string>

using namespace cv;

using namespace std;

class WatershedSegmenter {

private:

cv::Mat markers;

public:

void setMarkers(cv::Mat& markerImage)

{

markerImage.convertTo(markers, CV\_32S);

}

cv::Mat process(cv::Mat &image)

{

cv::watershed(image, markers);

markers.convertTo(markers, CV\_8U);

return markers;

}

}; //영상 마커 생성 및 정수형 영상 변환 그리고 워터쉐드 적용

int main(int argc, char\* argv[])

{

cv::Mat image = cv::imread("mun.jpg");

cv::Mat blank(image.size(), CV\_8U, cv::Scalar(0xFF));

cv::Mat dest;

imshow("original image", image);

// 마커 이미지 생성

cv::Mat markers(image.size(), CV\_8U, cv::Scalar(-1));

// x y 로 생성된 마커 이미지 활용

markers(Rect(0, 0, image.cols, 5)) = Scalar::all(1);

//상단부분

markers(Rect(0, image.rows - 5, image.cols, 5)) = Scalar::all(1);

//왼쪽부분

markers(Rect(0, 0, 5, image.rows)) = Scalar::all(1);

//오른쪽부분

markers(Rect(image.cols - 5, 0, 5, image.rows)) = Scalar::all(1);

//중간부분

int centreW = image.cols / 4;

int centreH = image.rows / 4;

markers(Rect((image.cols / 2) - (centreW / 2), (image.rows / 2) - (centreH / 2), centreW, centreH)) = Scalar::all(2);

markers.convertTo(markers, cv::COLOR\_BGR2GRAY);

imshow("markers", markers);

//워터쉐드 분할의 객체를 생성

WatershedSegmenter segmenter;

segmenter.setMarkers(markers);

cv::Mat wshedMask = segmenter.process(image);

cv::Mat mask;

convertScaleAbs(wshedMask, mask, 1, 0);

double thresh = threshold(mask, mask, 1, 255, THRESH\_BINARY);

bitwise\_and(image, image, dest, mask);

dest.convertTo(dest, CV\_8U);

imshow("final\_result", dest);

cv::waitKey(0);

return 0;

}