

Computer Vision Assignment Report

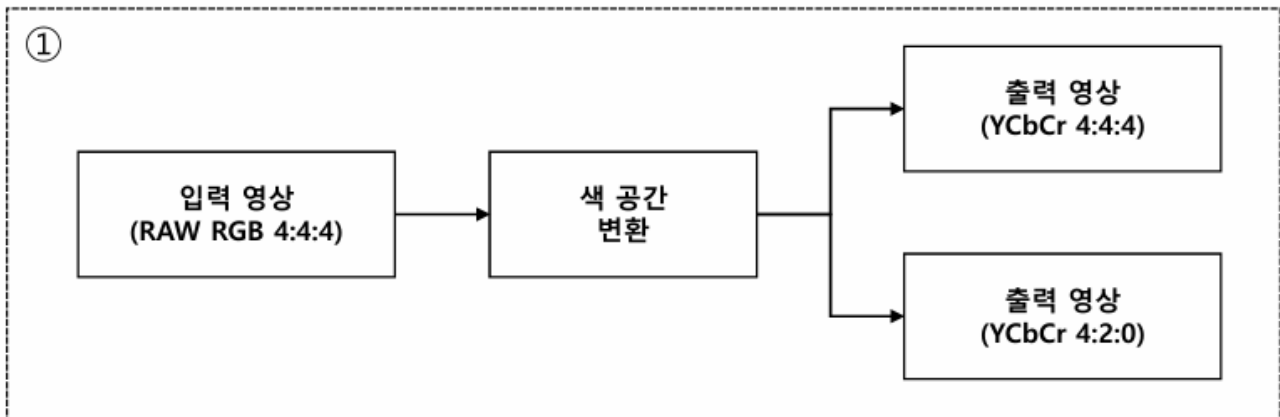
Assignment 1

2019742071 – 이민규

1. 과제 개요

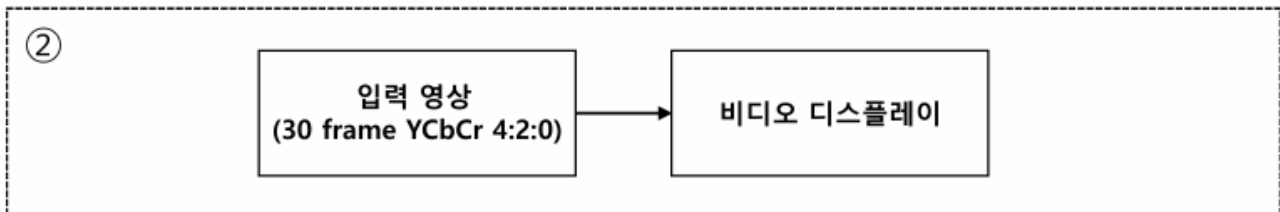
과제1: RGB 4:4:4 컬러 포맷 영상을 YCbCr 4:4:4와 YCbCr 4:2:0 컬러 포맷 영상으로 변환하는 프로그래밍 수행

- RGB 데이터 포맷과 YCbCr 데이터 포맷 간 변환하는 과정은 OpenCV 함수 사용하지 않고, 직접 구현
- RGB와 YCbCr의 각 채널 영상과 컬러 영상을 opencv의 imshow()함수를 이용하여 화면에 출력하고, 보고서에 첨부
- 변환된 YCbCr 4:4:4 & 4:2:0 데이터를 .yuv 파일로 저장



과제2: YCbCr 4:2:0 동영상을 입력 받아 화면에 출력하는 프로그래밍 수행

- YUV 4:2:0 동영상을 입력으로 받아 각 프레임 당 OpenCV의 imshow() 함수를 사용하여 화면에 출력 수행
- 변환된 30 frame RGB 데이터를 .raw 파일로 저장



2. 과제 수행 방법

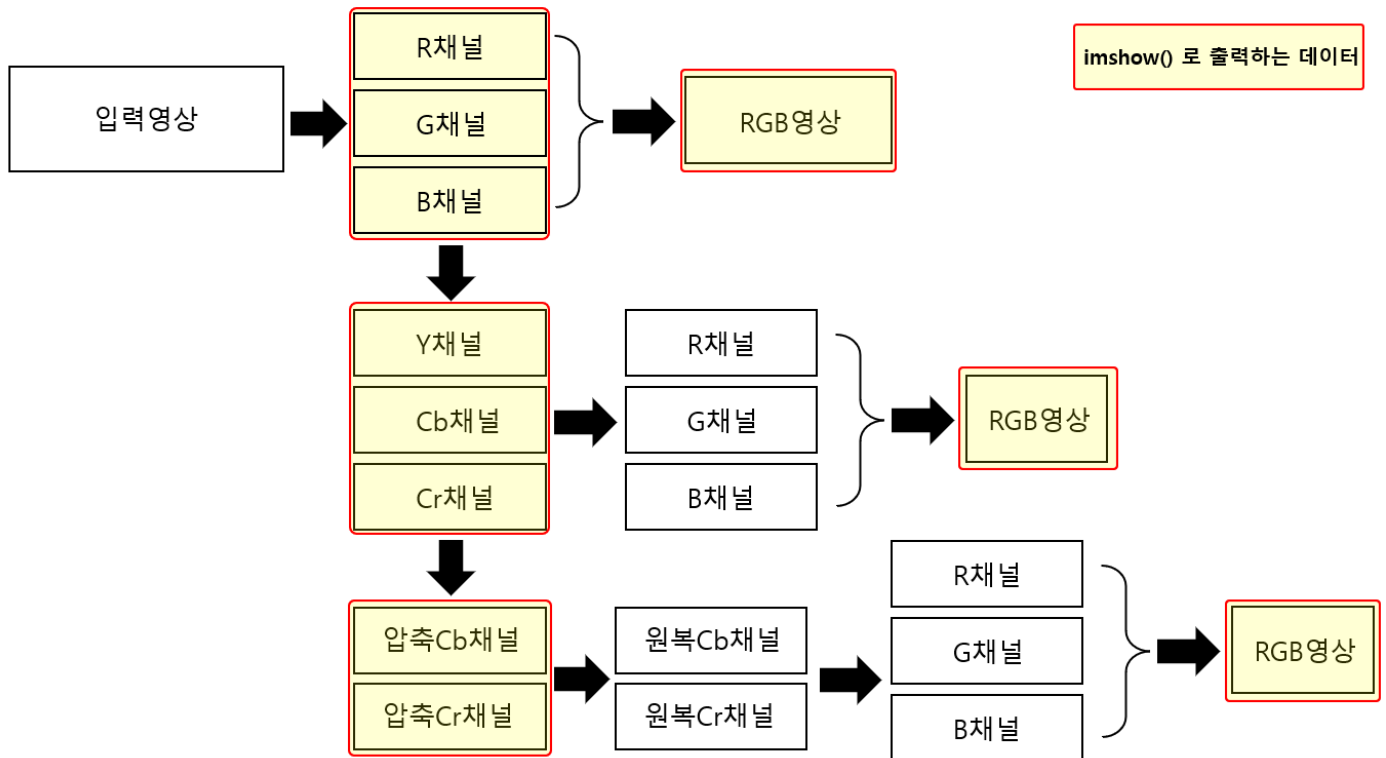


그림1. 과제1 수행코드의 수행도

과제1: 입력영상으로부터 R, G, B채널의 데이터를 각각 추출한 이후 직접 합쳐서 RGB이미지를 만든다. 각 채널의 데이터를 합칠 때 사용하는 객체는 Vec3b 객체로, 하나의 좌표에 3가지 채널의 데이터를 저장 가능하다.

R, G, B 채널의 데이터로 Y, Cb, Cr 채널로 색공간변환을 할 수 있으며 이때 <그림2>의 공식을 사용한다.

– RGB (8-bit) to YCbCr (8-bit)

$$\begin{aligned}
 Y &= 0.299R + 0.587G + 0.114B \\
 Cb &= 128 - 0.169R - 0.331G + 0.500B \\
 Cr &= 128 + 0.500R - 0.419G - 0.0813B
 \end{aligned}$$

– YCbCr (8-bit) to RGB (8-bit)

$$\begin{aligned}
 R &= 1.000Y + 1.402(Cr - 128) \\
 G &= 1.000Y - 0.714(Cr - 128) - 0.344(Cb - 128) \\
 B &= 1.000Y + 1.772(Cb - 128)
 \end{aligned}$$

그림2. RGB to YCbCr, YCbCr to RGB 변환 공식

변환한 Y, Cb, Cr 채널은 다시 RGB형식으로 변환하여 출력하고, Cb, Cr채널의 데이터를 압축하여 YCbCr 4:2:0 형태의 데이터로 만든다. 데이터를 압축할 때, 원본의 2X2영역의 픽셀 값의 평균을 새로운 압축데이터의 픽셀 값으로 쓰는 것이다. 이는 실제 OpenCV의 내장함수인 `resize()` 함수의 `INTER_LINEAR` 방식에서 착안하였다.

이에 따라 YCbCr 4:2:0은 Y채널의 크기는 그대로 유지하면서 Cb, Cr채널의 크기는 각각 1/4로 감소하여 총 절반의 데이터를 압축한다.

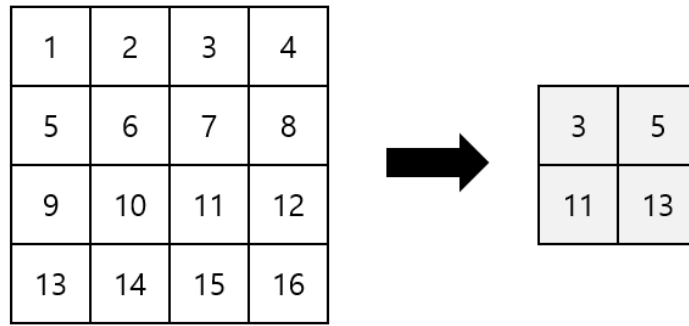


그림3. 선형보간법예시

압축한 데이터를 다시 원상복구할 때는 하나의 값을 2X2의 픽셀에 모두 복사하는 방법으로 복구하였다.

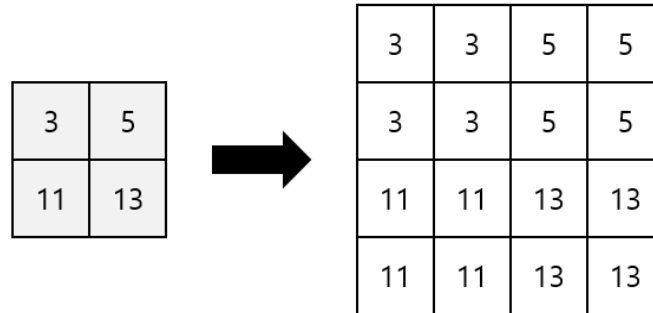


그림4. 원상복구예시

복구한 CbCr채널을 기반으로 다시 YcbCr을 RGB채널로 변환하여 RGB이미지를 얻을 수 있다.

위와 같은 방법으로 얻은 이미지데이터들을 imshow()함수를 사용하여 총 12개의 이미지를 출력한다.

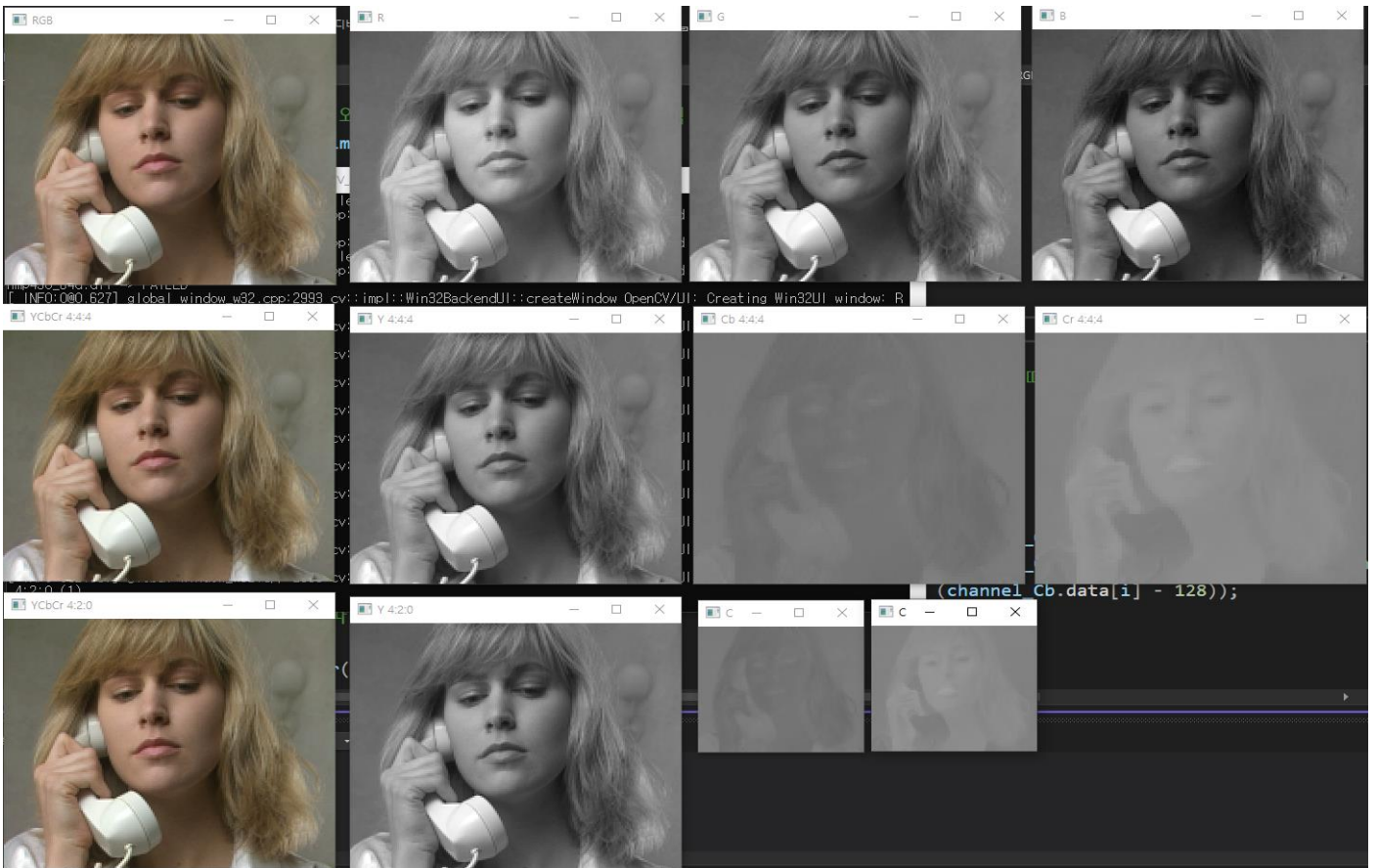


그림5. 출력결과

YUV 4:4:4 데이터와 YUV 4:2:0 데이터는 Y채널, Cb채널, Cr채널별로 나누어서 순차적으로 write한다.

이 파일은 YCbCr_420.yuv와 YCbCr_444.yuv 라는 이름으로 프로젝트 폴더에 저장되었다.

```
yuv444.write(reinterpret_cast<char*>(channel_Y.data), WIDTH * HEIGHT);
yuv444.write(reinterpret_cast<char*>(channel_Cb.data), WIDTH * HEIGHT);
yuv444.write(reinterpret_cast<char*>(channel_Cr.data), WIDTH * HEIGHT);
yuv444.close();

yuv420.write(reinterpret_cast<char*>(channel_Y.data), WIDTH * HEIGHT);
yuv420.write(reinterpret_cast<char*>(channelCb_420.data), WIDTH * HEIGHT / 4);
yuv420.write(reinterpret_cast<char*>(channelCr_420.data), WIDTH * HEIGHT / 4);
yuv420.close();
```

이름	수정한 날짜
x64	2024-03-18 오후 9:58
2019742071_이민규_CV1.cpp	2024-04-07 오후 6:44
Assignment1.vcxproj	2024-04-07 오후 6:44
Assignment1.vcxproj.filters	2024-04-07 오후 6:44
Assignment1.vcxproj.user	2024-03-18 오후 9:28
opencv_world490.dll	2023-12-28 오전 9:16
opencv_world490d.dll	2023-12-28 오전 9:10
output.raw	2024-04-07 오후 6:52
RaceHorses_416x240_30.yuv	2024-03-18 오후 5:56
Suzie_CIF_352x288.raw	2024-03-18 오후 5:56
YCbCr_420.yuv	2024-04-07 오후 6:51
YCbCr_444.yuv	2024-04-07 오후 6:51

그림6. 프로젝트 폴더에 저장된 yuv파일

과제2: 입력영상으로부터 읽는 데이터는 Y, 압축된 CbCr채널이다. Cb, Cr채널은 원본의 1/4크기를 준비한다.

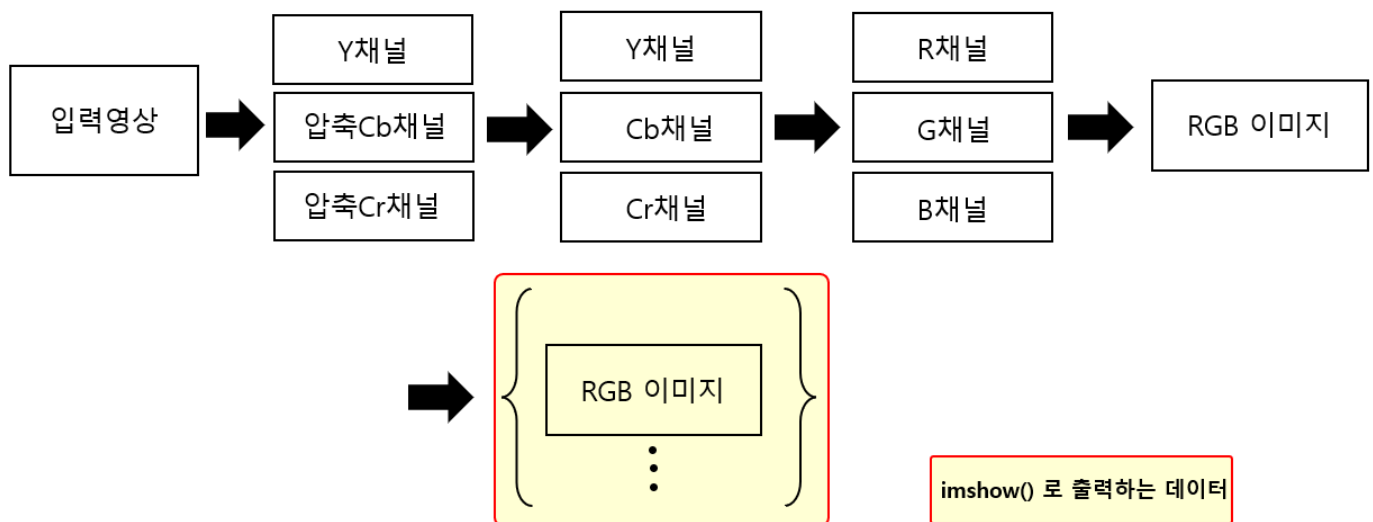


그림7. 과제2 수행코드의 수행도

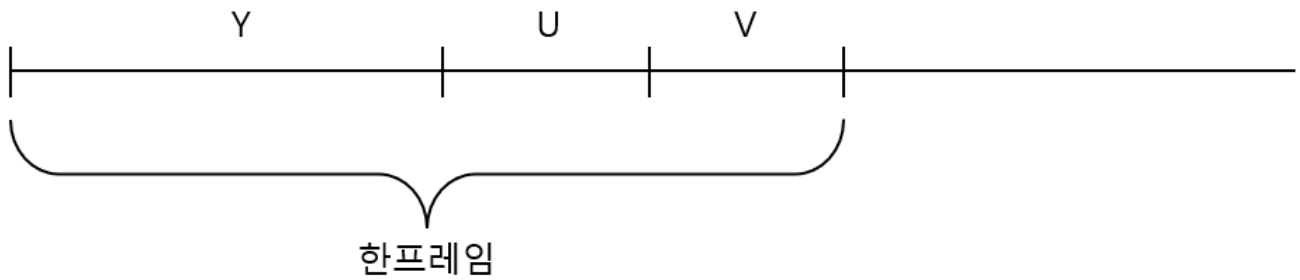


그림8. file객체에 저장된 데이터의 형태

File에 저장된 원본파일을 1차원의 메모리의 형태로 표현하면 위와 같다. 반복문이 한번 실행될 때마다 1프레임을 버퍼에 저장하는데 하나의 프레임에는 Y채널, U채널, V채널 데이터가 순차적으로 저장되어있다.

과제1을 수행할 때, YCbCr 4:2:0 데이터에서 CbCr채널만 따로 원본크기로 사이즈업 하였다. 동일한 방식으로 YCbCr채널을 얻는다. 얻은 YCbCr채널을 <그림2>의 공식을 사용하여 RGB채널로 변경한다. 각각의 RGB채널을 합쳐서 RGB이미지를 얻으면 한번의 반복문 실행마다 각 이미지를 일정 시간의 간격으로 출력하여 영상을 재생할 수 있다.



그림8. 과제2 출력결과

```
for (int i = 0; i < one_frame.rows; ++i)
    raw.write(reinterpret_cast<char*>(one_frame.ptr(i)), one_frame.cols * one_frame.elemSize());
```

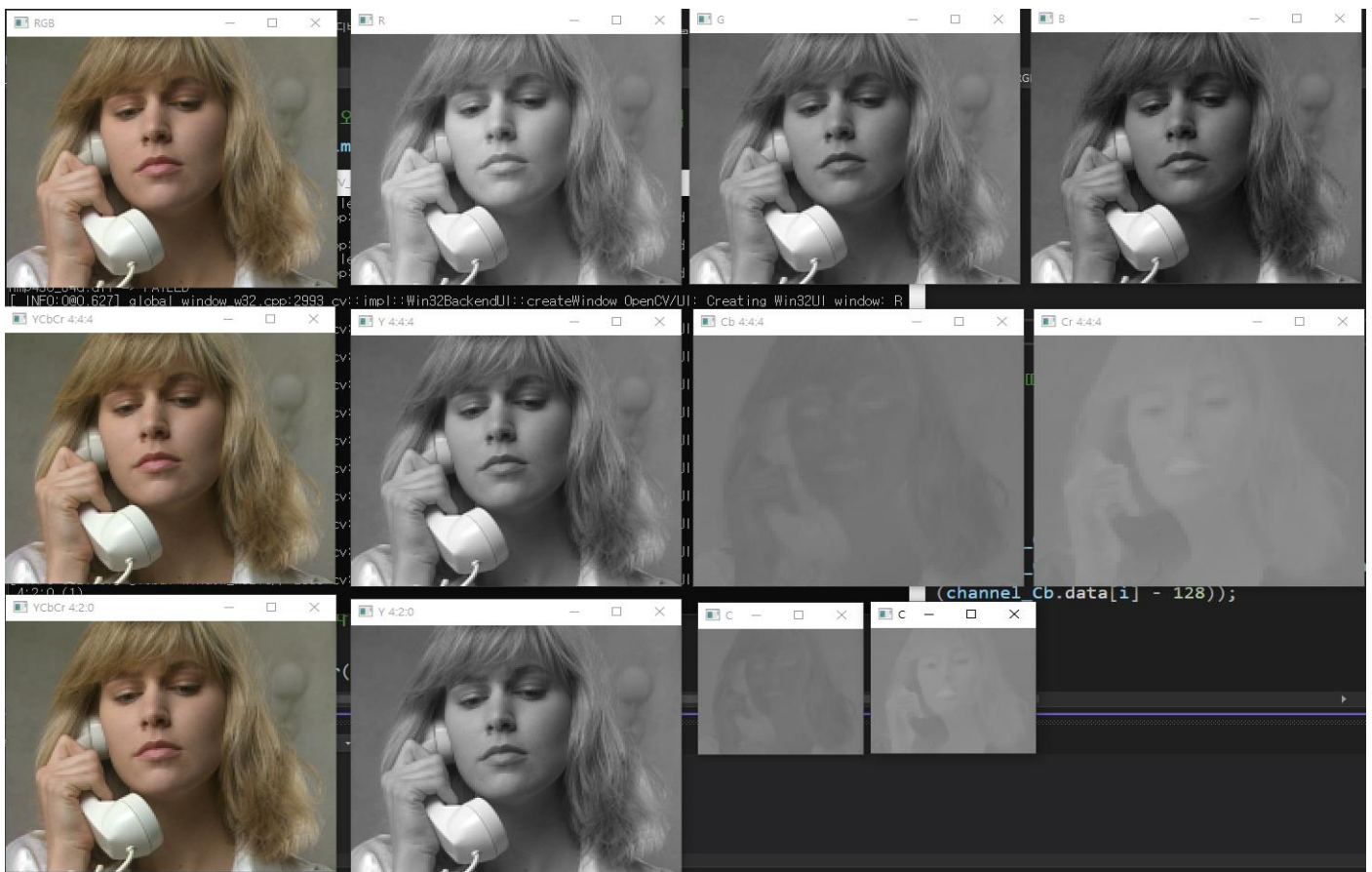
위 코드를 사용하여 반복문 1회당 한프레임씩 주어진 파일에 write한다. 한 프레임은 다시 각 행으로 나뉘어 순차적으로 파일에 작성된다. 이 파일은 output.raw라는 이름으로 프로젝트 폴더에 저장되었다.

Assignment1		Assignment1
이름		수정한 날짜
x64		2024-03-18 오후 9:58
2019742071_이민규_CV1.cpp		2024-04-07 오후 6:44
Assignment1.vcxproj		2024-04-07 오후 6:44
Assignment1.vcxproj.filters		2024-04-07 오후 6:44
Assignment1.vcxproj.user		2024-03-18 오후 9:28
opencv_world490.dll		2023-12-28 오전 9:16
opencv_world490d.dll		2023-12-28 오전 9:10
output.raw		2024-04-07 오후 8:58
RaceHorses_416x240_30.yuv		2024-03-18 오후 5:56
Suzie_CIF_352x288.raw		2024-03-18 오후 5:56
YCbCr_420.yuv		2024-04-07 오후 8:58
YCbCr_444.yuv		2024-04-07 오후 8:58

그림9. 프로젝트 폴더에 저장된 raw파일

3. 결과분석

과제1



각 채널의 출력결과가 주어진 예시와 동일하게 출력됨을 볼 수 있다. RGB채널, YCbCr 4:4:4, YCbCr 4:2:0의 결합으로 만든 RGB이미지도 색공간변환, 압축과 복구로 기존 원본과 그 데이터의 구성이 달라졌음에도 불구하고 육안으로는 차이를 구분하기 어렵다.

YCbCr_420.yuv
YCbCr_444.yuv

2024-04-07 오후 8:58
2024-04-07 오후 8:58

YUV 파일
YUV 파일



149KB
297KB

YCbCr 4:2:0 압축파일과, YCbCr 4:4:4 원본의 저장된 YUV파일을 확인하면 그 용량이 각각 149KB, 297KB로 이론과 같이 YCbCr 4:2:0로 압축했을 때 원본에 비해 그 용량이 절반으로 감소함을 확인 가능하다.

과제2



변환한 영상이 문제없이 출력됨을 확인하였다

 output.raw	2024-04-07 오후 8:58	RAW 파일	88,043KB
 RaceHorses_416x240_30.yuv	2024-03-18 오후 5:56	YUV 파일	43,875KB

RaceHorses 원본파일은 YCbCr 4:2:0 파일로, 압축이 되어있는 상태이다. 이를 RGB이미지로 변형하여 저장한 output.raw 파일은 88,042KB로 43,875KB의 대략 두배크기임을 확인할 수 있다.

4. 고찰

이번 구현은 간단하게 imread()로 분석가능한 jpg나 png와는 다르게 모든 데이터를 직접 file에 읽어내야하는 raw파일을 다루어 색공간변환과 출력을 구현하였다. 실제 내가 생각하고 있던 raw파일의 저장방식이 차이가 있어 여러 번의 시행착오를 겪으면서 파일을 읽는 시도를 하였다. 컬러 raw파일을 읽는 것의 핵심은 처음부터 Vec3b 객체로 읽는 것이 아니라 파일 내부의 데이터를 고려하여 이를 R채널, G채널, B채널로 분리하는 것이다.

헤더파일이 없으므로 사용자가 raw파일의 해상도를 알고 있어야 해석할 수 있다는 점도 큰 단점이다. 용량이 작다는 점 외에 raw파일의 장점이 없어보이는 만큼 컴퓨터비전 현업분야에서 raw파일을 사용한다면 그 이유를 추후에 살펴보는 것도 좋은 탐구주제가 될 것으로 생각된다.

RGB데이터를 YCbCr 4:4:4와 YCbCr 4:2:0으로 변환시켰으며 그 반대 또한 구현하였다. 이를 출력하였을 때, 실제 육안상으로는 각각의 이미지를 구분할 수는 없었다. YCbCr 4:2:0 포맷은 데이터의 용량을 획기적으로 줄이면서 감당가능한 해상도를 유지하는 좋은 방법인 것으로 생각된다. 본 구현에서는 2X2 픽셀의 평균값으로 압축하고, 다시 사이즈를 늘릴 때는 하나의 픽셀값을 2X2 픽셀에 그냥 복사하는 방법을 사용하였다. 이보다 좀더 시간이나 메모리 자원이 소모된다 하더라도 더 높은 정확성의 복구방법이 사용되고 있을 것으로 생각되는데, 추후 이러한 방법을 조사한다면 더 좋은 해상도의 영상을 얻을 수 있을 것으로 기대한다.