

w11 화소단위 처리 & 화질개선

202114160 채민기

개요

1. page4~6 화소 접근 방법에 따른 시간차이 확인

1. 사이즈가 2000x2000일 때
2. 사이즈가 1000x1000일 때
3. 사이즈가 320x240일 때
 1. 사이즈 320x240원본
 2. 사이즈 320x240 이미지를 2000x2000으로 resize
 3. 사이즈 320x240 이미지를 1000x1000으로 resize

결과표 및 최종 결론

2. page8~9 의 chromakey효과 실행

검은색의 배경을 변경
초록색의 배경을 변경

3. page14의 코드를 실행

- 1.R성분이 많은 영상
 - 2.G성분이 많은 영상
 - 3.B성분이 많은 영상
- 추가실험 P와, Y성분의 영상을 하면 어떻게 나오는가?

4.컬러 영상에 대한 histogram equalization을 구현


- 1.BGR요소에 직접 equalization
- 2.YCrCb에 대해 equalization

1. page4~6 화소 접근 방법에 따른 시간차이 확인

이미지의 크기가 작다면 4가지 방법 모두 빠르게 완료가 될 것이므로
원본이 큰 이미지 2000x2000부터 1000x1000, 그리고 수업시간 자주 사용하는
크기인 320x240까지 원본의 이미지 크기가 차이나는 3개의 이미지를 가지고 실험을 진행
하였다.

가장 고화질인 2000x2000의 이미지의 경우 화소처리의 시간리 가장 오래 걸릴 것이기 때
문에 가장 오래걸리는 화소접근 방식인 array방식에서 큰 차이를 확인 할 수 있을 것이라
생각된다.

다음의 사진은 사용된 사진들의 원본 픽셀수다.

 320x240	2024-05-17 오전 12:09	JPG 파일
 1000x1000	2024-05-17 오전 12:07	JPG 파일
 2000x2000	2024-05-17 오전 12:03	JPG 파일

이미지		
이미지 ID		
사진 크기	2000 x 2000	
너비	2000픽셀	
높이	2000픽셀	

이미지		
이미지 ID		
사진 크기	1000 x 1000	
너비	1000픽셀	
높이	1000픽셀	
수업 해상도	200 DPI	

이미지 ID		
사진 크기	320 x 240	
너비	320픽셀	
높이	240픽셀	

1. 사이즈가 2000x2000일 때 <가장 많은 화소>

```
PS D:\code\mmp\w11> & C:/Users/vtoree/AppData/Local/Programs/Python/Python39-64/Scripts/python.exe or_report/4~6.py
Enter option (1=2000, 2=1000, or 3=320x240): 1
Elapsed time for method 1: 4.69936371 seconds
Elapsed time for method 2: 0.75749207 seconds
Elapsed time for method 3: 0.39680266 seconds
Elapsed time for method 4: 0.00000000 seconds
PS D:\code\mmp\w11> & C:/Users/vtoree/AppData/Local/Programs/Python/Python39-64/Scripts/python.exe or_report/4~6.py
Enter option (1=2000, 2=1000, or 3=320x240): 1
Elapsed time for method 1: 4.64260864 seconds
Elapsed time for method 2: 0.74644518 seconds
Elapsed time for method 3: 0.36155725 seconds
Elapsed time for method 4: 0.01562331070 seconds
```

<2000x2000의 이미지에 대한 화소접근 방법별 시간차>

처음 소수점표기를 8자리까지 하였음에도 가장 빠른 화소접근 방식인 4번째 방식의 경우 0.000000으로 표기되어 12자리로 수정후 확인 할 수 있었다.

time.time()이 매우 짧은 시간이라면 반올림하여 0초로 표기 될 수 있다고 하여 **time.perf_counter()**을 통해 좀더 정밀하게 시간계산을 진행하였다.

```
start_time = time.perf_counter()
img4 = gammaCorrect4(gray, gamma)
print(f"Elapsed time for method 4: {time.perf_counter() - start_time:.8f} seconds")
start_time = time.perf_counter()
```

```
or_report/4~6.py
Enter option (1=2000, 2=1000, or 3=320x240): 1
Elapsed time for method 1: 4.68396091 seconds
Elapsed time for method 2: 0.77457714 seconds
Elapsed time for method 3: 0.38327193 seconds
Elapsed time for method 4: 0.00825420 seconds
PS D:\code\mmp\w11>
```

<시간 관련 코드 수정후 동일한 이미지에 대한 처리시간>

가장 느린 array접근 방식은 4.6초 정도 였으며 Look up table과 numpy array의 인덱스를 대치시킨 4번째 접근 방식은 0.015~0.008초 정도로 약 563배의 속도차이 가 발생하였으며 item접근과는 약 7배 정도의 차이가 발생하였다.

즉 가장 느린 1번 방법과의 속도차 약 7배,14배, 563배로 큰 차이를 보였다.

2. 사이즈가 1000x1000일 때

```
PS D:\code\mmp\w11> & C:/Users/vtoree/AppData/Local/Programs/Python/Python39-64/Scripts/python.exe C:/Users/vtoree/AppData/Local/Programs/Python/Python39-64/Scripts/python.exe or_report/4~6.py
Enter option (1=2000, 2=1000, or 3=320x240): 2
Elapsed time for method 1: 1.16292930 seconds
Elapsed time for method 2: 0.19209218 seconds
Elapsed time for method 3: 0.09867668 seconds
Elapsed time for method 4: 0.00220950 seconds
PS D:\code\mmp\w11>
```

<1000x1000의 이미지에 대한 화소접근 방법별 시간차>

1000x1000의 경우 2000x2000의 이미지와 픽셀차가 4배의 차이가 존재 한다.
따라서 해당 이미지의 처리시간들 역시 4가지 화소 접근 방식 모두 4배의 시간차이가 발생하였다. 따라서 가장 느린 array접근 방식은 가장 빠른 4번째 화소 접근 방식은 2000x2000과 유사하게 약 528배의 속도차이가 발생하였으며 item을 이용한 접근 방식과는 약 6배의 시간차이가 발생하였다.
즉 가장 느린 1번 방법과 약 6배, 13배, 528배로 큰 차이를 보였다.

3. 사이즈가 320x240일 때 <가장 적은 화소>

```
PS D:\code\mmp\w11> & C:/Users/vtoree/AppData/Local/Programs/Python/Python39-64/Scripts/python.exe C:/Users/vtoree/AppData/Local/Programs/Python/Python39-64/Scripts/python.exe or_report/4~6.py
Enter option (1=2000, 2=1000, or 3=320x240): 3
Elapsed time for method 1: 0.10669994 seconds
Elapsed time for method 2: 0.01931691 seconds
Elapsed time for method 3: 0.00761032 seconds
Elapsed time for method 4: 0.00027230 seconds
PS D:\code\mmp\w11>
```

<320x240의 이미지에 대한 화소접근 방법별 시간차>

가장 크기가 작은 이미지에 대한 화소접근을 진행하였으므로 4가지 화소접근 방법 모두 1초가 안되는 매우 짧은 시간에 이미지처리가 완료되었다.
2000x2000의 이미지와 픽셀수가 약 52배 차이가 있었으며 4가지 방법모두 이와 약 40배의 차이로 빠르게 처리된 것을 확인 할 수 있다.
모든 화소 접근 방식이 빠른 속도를 보였지만 이번에도 가장 느린 array 접근 방법과 비교할 때 각각 약 5.6배 14배 394배의 여전히 큰 속도차이를 확인 할 수 있었다.

추가실험

```
Elapsed time for method 4: 0.00822200 seconds
PS D:\code\mmp\w11> & C:/Users/vtoree/AppData/Local/Programs/Python/Python38-32/Python.exe or_report/4~6.py
Enter option (1=2000, 2=1000, or 3=320x240): 3
Select image size:
1: 2000x2000
2: 1000x1000
3: 320x240
Enter option (1, 2, or 3): 1
Elapsed time for method 1: 4.68224072 seconds
Elapsed time for method 2: 0.75972819 seconds
Elapsed time for method 3: 0.35662818 seconds
Elapsed time for method 4: 0.00846810 seconds
PS D:\code\mmp\w11> █
```

<320x240의 이미지를 2000x2000으로 resize후의 화소접근 방법별 시간차>

```
Elapsed time for method 4: 0.00846810 seconds
PS D:\code\mmp\w11> & C:/Users/vtoree/AppData/Local/Programs/Python/Python38-32/Python.exe or_report/4~6.py
Enter option (1=2000, 2=1000, or 3=320x240): 3
Select image size:
1: 2000x2000
2: 1000x1000
3: 320x240
Enter option (1, 2, or 3): 2
Elapsed time for method 1: 1.17260504 seconds
Elapsed time for method 2: 0.18586946 seconds
Elapsed time for method 3: 0.09011102 seconds
Elapsed time for method 4: 0.00228780 seconds
PS D:\code\mmp\w11> █
```

<320x240의 이미지를 1000x1000으로 resize후의 화소접근 방법별 시간차>

가장 작은 이미지인 320x240을 resize를 통해 이미지 크기를 2000x2000과 1000x1000의 크기로 수정한후 코드를 실행하여도 원본의 크기가 2000x2000 일 때와 1000x1000일 때 소요되는 시간과 오차가 거의 발생하지 않았다.

결과표 및 최종 결론

	array접근	item접근	LUT+item접근	LUT+array
2000x2000	4.6839	0.7745	0.3232	0.0082
1000x1000	1.1629	0.1920	0.0986	0.0022
320x240	0.1066	0.0193	0.0076	0.0002

<이미지 크기와 화소 접근 방법별 시간차 (소수점 아래 4자리까지 표기) >

즉 array를 통한 화소 접근 방식이 그 다음으로 느린 item을 이용한 접근 방식보다 최소 5배이상 느렸으며 가장 빠른 방법과도 약 400배 이상의 시간차이가 발생하는 것을 확인 할 수 있었다.

array를 통한 접근 보다는 item을 통한 접근 방식을 사용하는 것이 여러 이미지를 한번에 처리할 때 좋으며, Look up table을 구할 수 있는 처리 방식이라면 Lut와 함께 array를 대치 시켜 화소처리를 진행하는 것이 시간적으로 가장 효율적임을 알 수 있다.

2. page8~9 의 chromakey효과 실행



<갤럭시 줌으로 찍은 달 사진>



<사용된 배경 지구 사진>

첫 번째 사진의 경우 배경이미지가 검은 하늘이기 때문에 낮은 값들의 성분을 수정해준다면 쉽게 달의 이미지를 합성할 수 있을 것 이라본다.

두 번째 사진은 풀숲의 배경부분을 백두산 천지 이미지 부분으로 변경하는 것이므로 G채널의 값만을 잘 조절해주면 좋은 이미지의 사진을 얻을 수 있을 것이라고 판단된다.

검은색의 배경을 수정



<RGB채널 값을 모두 50이하인 성분이 변경되도록 설정>

핸드폰을 통해 100배 줌으로 확대하여 찍었던 달 사진이기 때문에

주변의 배경이 동일한 검은 색상을 보였다. 따라서 달의 사진을 제외한 나머지 배경부분의 색을 원하는 배경화면과 합성하는 것이 어렵지 않았다.

일반적인 경우 크로마키는 뒤의 배경을 한가지 색으로 통일하며 초록색상을 많이 사용한다. 단색은 아니지만 초록색상을 띄고 있는 풀숲의 경우는 어떤 결과가 나올지 알고 싶어 추가 실험을 진행하였다.

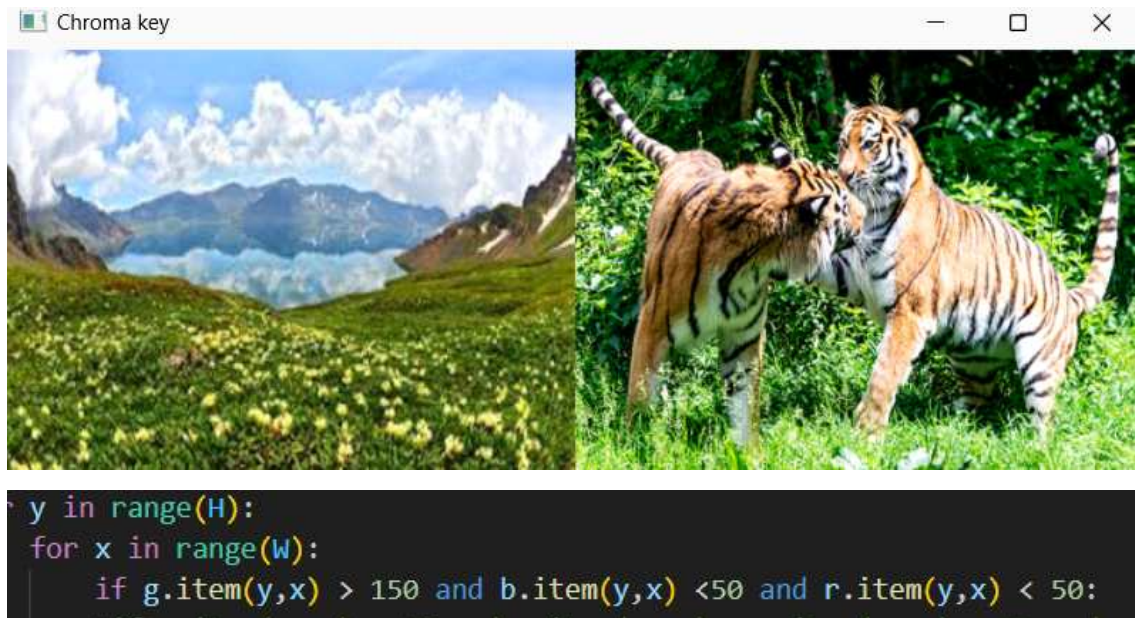


<사용된 호랑이 사진>



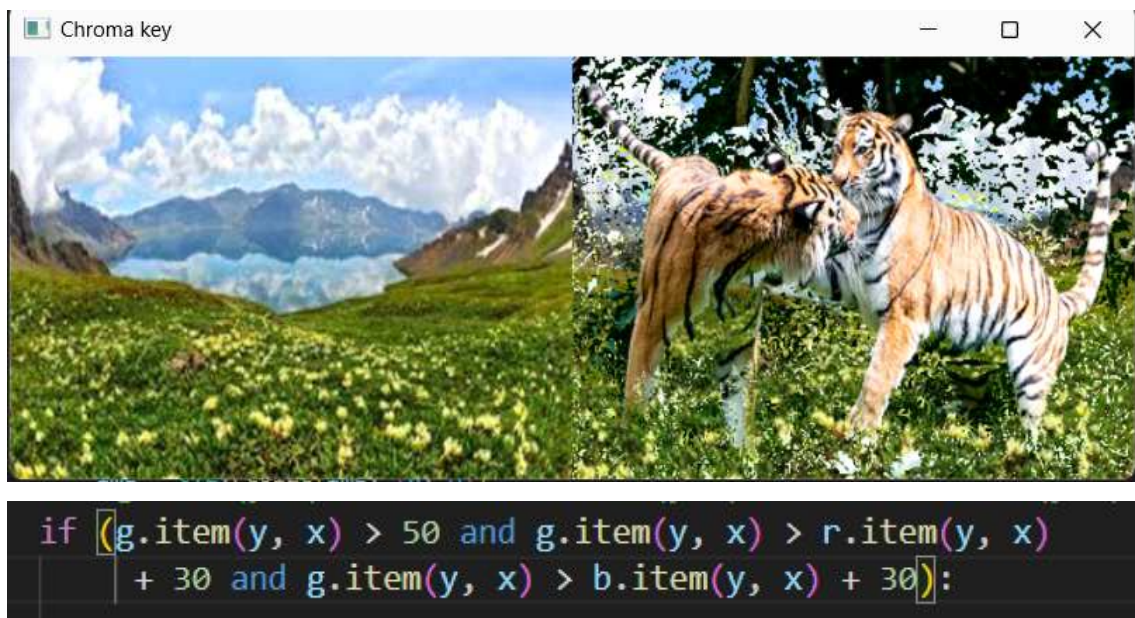
<사용된 배경 백두산 천지>

초록색의 배경을 수정



해당 코드는 G의 밝기가 높으며 R,B의 밝기는 상대적으로 낮은 픽셀 즉 초록색이 다른 두 채널에 비해 상대적으로 더 강조된 픽셀을 대체하려 했다. 하지만 이 조건이 초록색이 다른 두 색상에 비해 상당히 높을 때만 적용이 되기 때문에 다른 색상이 섞인 경우 필터링이 안 될 가능성이 있었다.

코드수정 후



조건을 다음과 같이 초록색이 50이상의 밝기를 가지면서 R과B채널보다 30이상 높

은 픽셀을 선택하도록 하였다.

이를 통해 초록색의 강도가 다른 색상보다 높은 경우에 변경이 가능토록하여 이미지가 이전에 비해 합성이 된 것을 볼 수 있다.

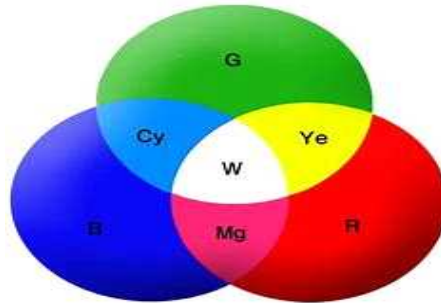
인위적으로 배경의 일부분 초록색으로 변경



초록색 부분이 적어 이미지가 잘 변경되지 않은 부분을 인위적으로 초록색 배경으로 변경해준뒤 크로마키 코드를 실행하였을 때 호랑이를 제외한 대부분의 영역이 배경사진과 잘 변경된 것을 확인 할 수 있다.

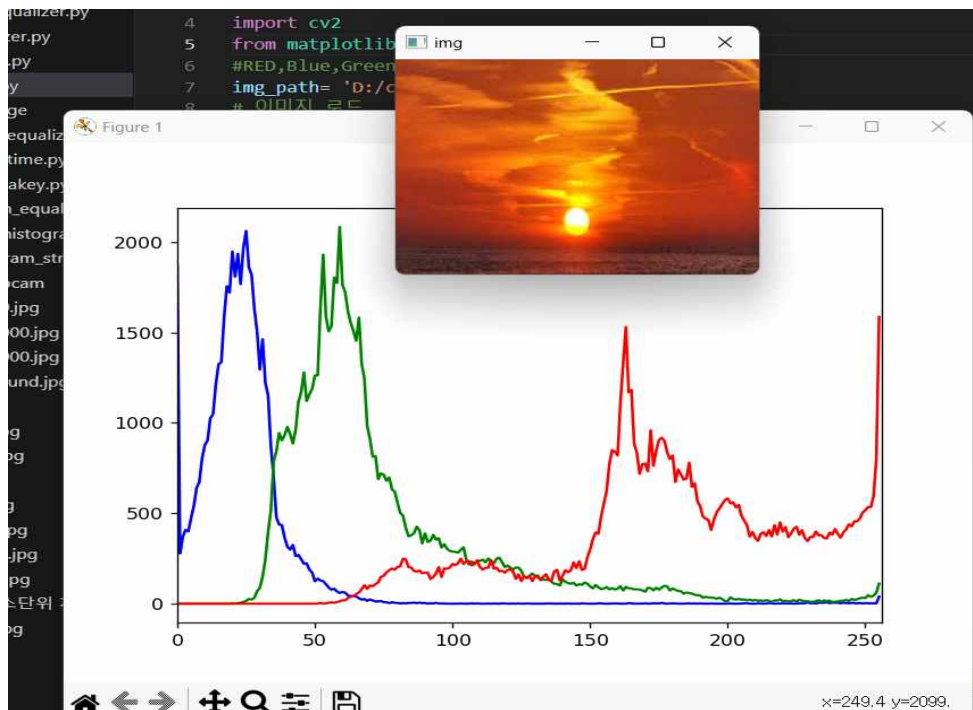
이를 통해 영화를 촬영할 때 크로마키를 위한 배경을 색의 변화가 없는 단색의 초록색 배경을 사용하는 이유와 해당 배경과 유사한 색상의 초록색이 포함된 옷이나 장비를 입지 않는 이유를 확인할 수 있었다.

3. page14의 코드를 실행



<빛의 합성에 따른 색상>

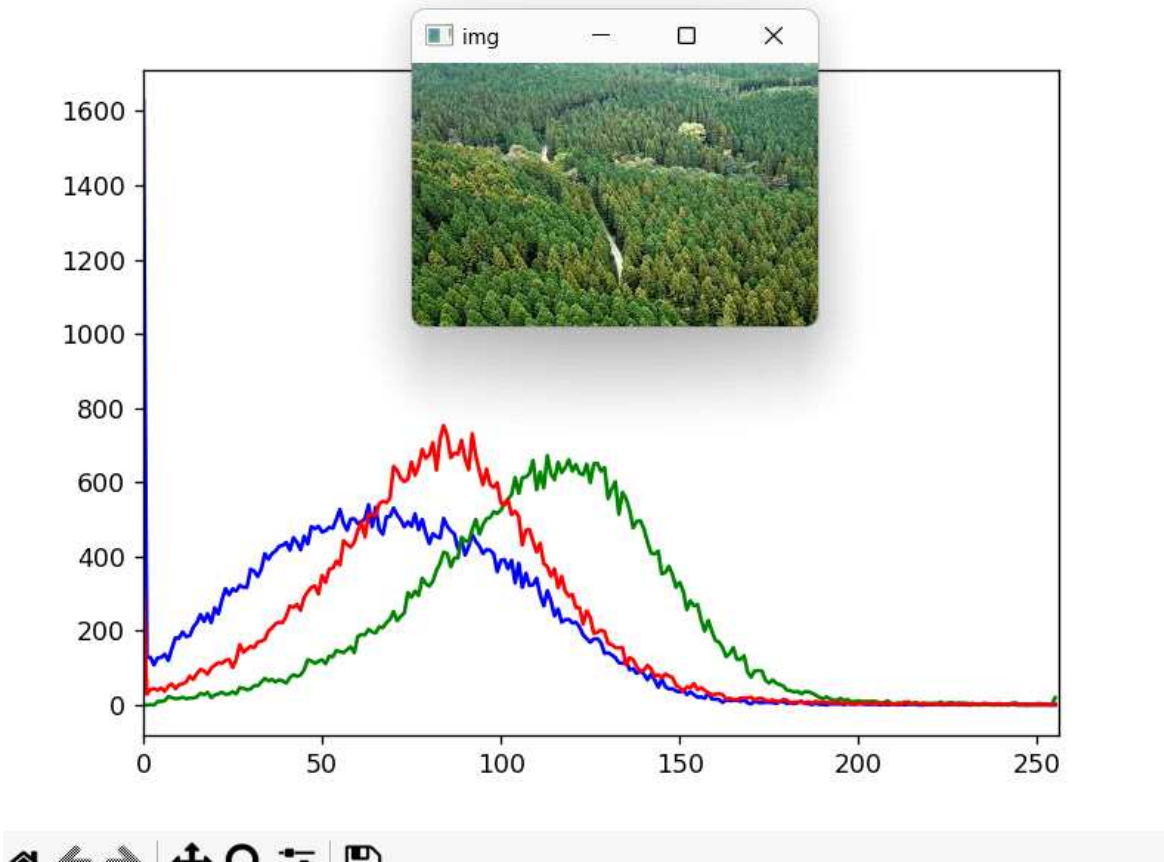
1.R성분이 많은 영상



<붉게 노을진 하늘>

빨간색인 R성분이 가장 많이 분포하는 영상으로 해질녘 하늘 사진을 사용하였다. 하늘이 붉게 물들어 예상대로 R채널에 해당하는 히스토그램이 큰 값을 나타내는 오른쪽에 많이 분포하는 것을 볼 수 있다. G채널과 B채널에 해당 하는 히스토그램은 비교적 낮은 값에 분포하는 것을 확인할 수 있었다. 이는 실제로 하늘에서 파란색이 많이 사라지고 R채널의 붉은 색의 색상이 우세함을 나타낸다. G채널의 값은 하늘과 태양주변의 빛이 주황색, 노란색, 흰색에 가까운 색상들로 인해 G채널의 성분이 어느정도 합성되어 있음을 의미한다.

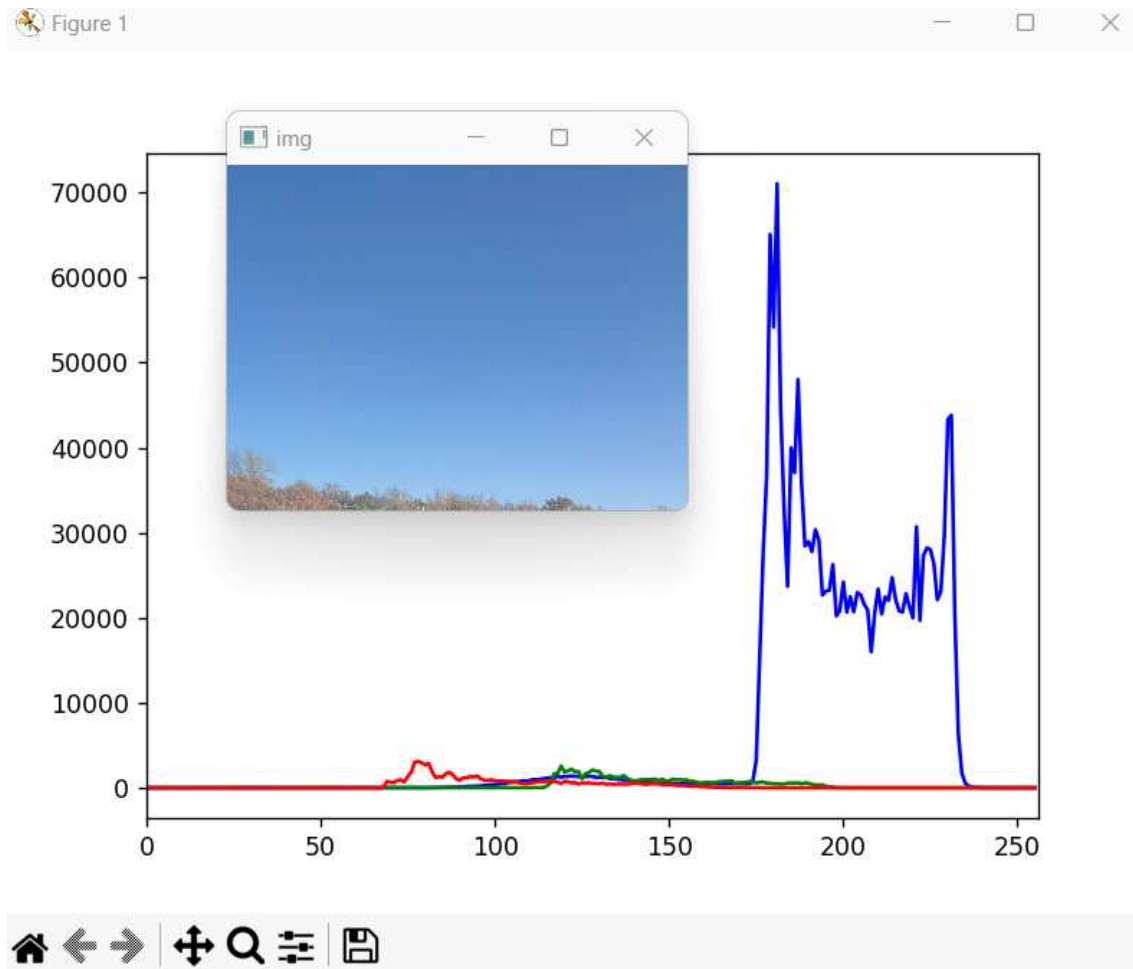
2.G성분이 많은 영상



<초록 빛을 띄는 산위에서 찍은 숲의 사진>

초록색에 해당하는 G채널의 성분이 가장 많은 영상은 숲의 사진을 선택하였다. 히스토그램을 확인하면 모든 채널의 색상이 골고루 분포하였지만 초록색에 해당하는 G채널의 히스토그램이 R, B채널에 비해 높은 값인 우측에 치우쳐져 있음을 알 수 있다. 이는 앞서 빛의 합성에 따른 색상에서 확인 할 수 있듯이 숲의 나무의 색이 단순한 초록색이 아닌 G와 B의 합성인 청록색과 G와 R의 합성인 노란빛깔에 가까운 연두색의 나뭇잎들로 인해 존재하기 때문이다.

3.B성분이 많은 영상



<전역날 하늘 사진>

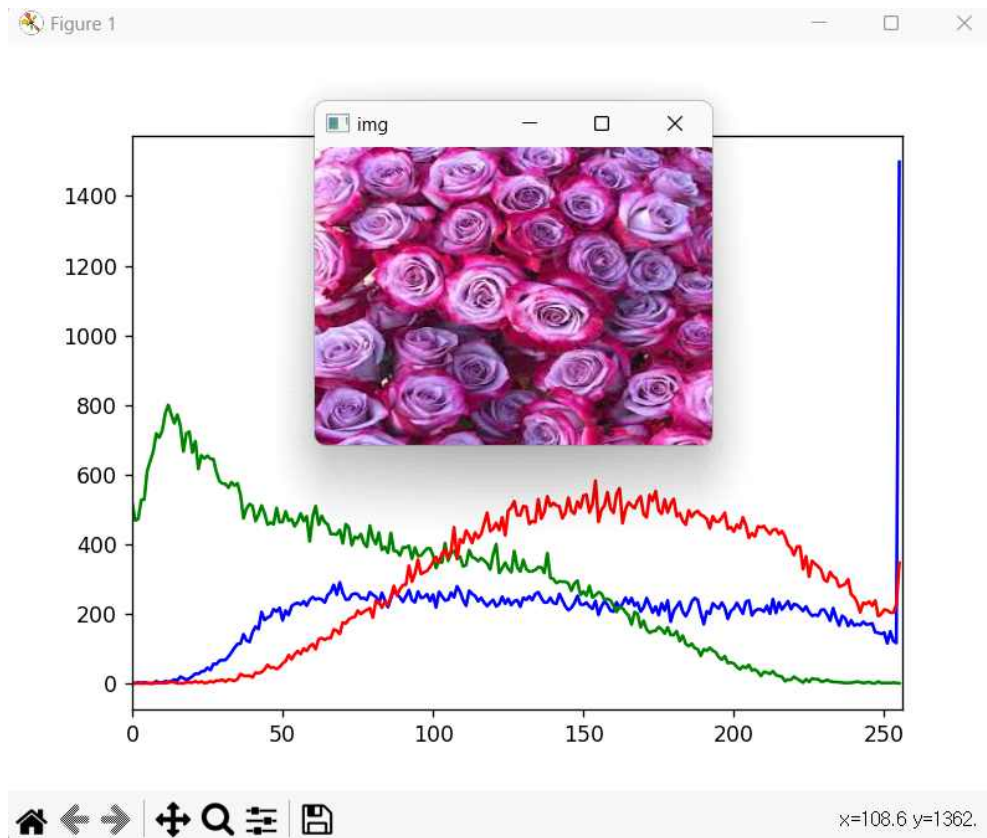
파란색에 해당하는 B채널의 성분이 가장 많은 영상은 하늘 사진을 사용하였다. 히스토그램을 파란색의 B채널의 히스토그램이 높은 값에서 큰 피크를 보여주는 것을 확인 할 수 있다. 이는 하늘의 파란 색이 히스토그램에 반영된 것을 의미한다. 붉은 색 성분과 초록색 성분은 아래 일부가 찍혀 있는 가을나무들로 인해 어느 정도 분포하였으나 미미한 것을 확인 할 수 있다.

추가실험 P와, Y성분의 영상을 하면 어떻게 나오는가?

앞선 실험으로 R,G,B성분이 각 색상들이 합성된 색으로 인해 예상보다 검출이 되는 경우가 존재하였다.

R과G를 합성한 Y색상 그리고 R과 B를 합성한 MG(자홍색)의 사진을 사용한다면 각 두가지 색이 합성에 사용되지 않은 한가지 색보다 우세한지를 실험을 통해 알아 보려한다.

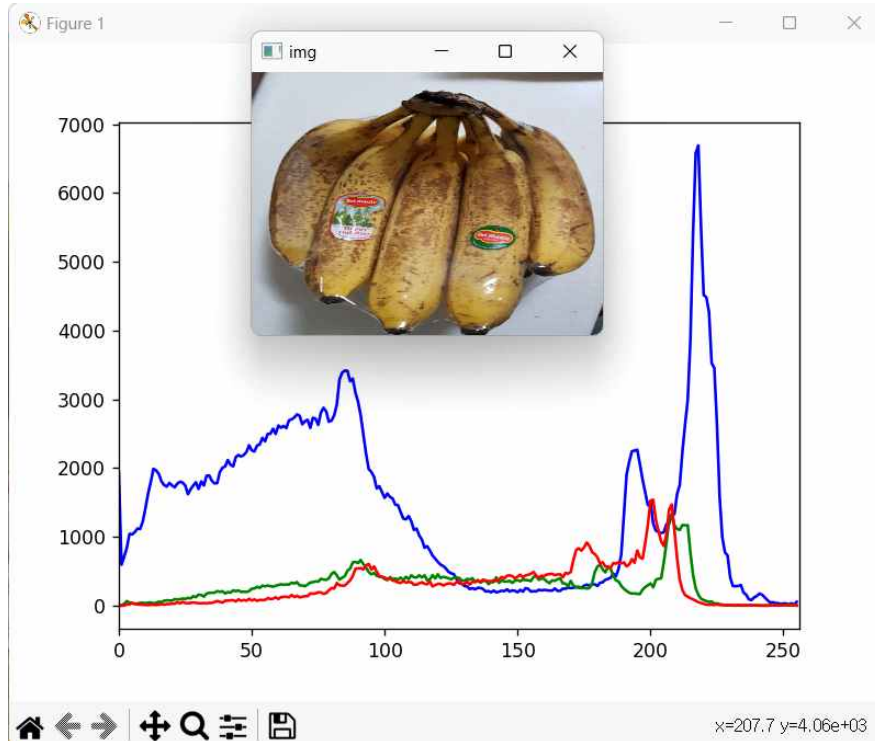
4.mg성분이 많은 영상



<자홍색 장미사진>

자홍색은 R과 B의 합성으로 만들어진다. 사진은 자홍색 장미를 사용하였다. G성분에 비해 R과B가 높은 값에 치우쳐져 있으며 200 이상의 큰 값일수록 차이가 두드러진다. Mg성분이 R과B의 성분이 주성분이지만 G성분역시 어느정도 분포하는 것을 확인 할수 있다. 이는 장미의 자홍빛 색깔이 다채롭게 연보라 빛을 띄기도 하는데 이는 자홍색 성분에 흰색 성분이 추가되어 옅은 빛깔을 띄게 하는 것이다. 따라서 이에G성분이 필요하여 G성분도 검출이 된것이라 볼 수 있다.

5. Y성분이 많은 영상



<노란색 바나나사진>

노란색 성분을 검출하기 위해 바나나사진을 사용하였다.

노란색 성분은 R과 G의 합성으로 만들어지므로 B성분은 배경의 흰색에서 일부가 검출 될것이라 생각하였으나 가장 많이 검출된 성분이 B성분이였다.

R과 G성분은 Y성분 합성으로 인해 거의 동일하게 증가하는 그래프를 가지고 있다.

이로인해 Y성분은 G와 R성분의 합성으로 인한 것이라 볼 수 있었다.

B성분이 많이 검출된 이유중 하나는 바나나의 갈변 효과로 인해 발생한 갈색 부분으로 인한 현상이다.

R:254 G:190 B:21	R:69 G:48 B:27	R:113 G:84 B:50
------------------------	----------------------	-----------------------

<갈색 계열의 RGB 값>

또한 갈색 계열에서 B성분 역시 낮은 값에 포함된 것을 알 수 있다.

히스토그램에서 B성분이 가장 낮은 값에 많이 분포하는 이유가 배경의 갈색부분과 그릇의 흰색으로 인한 것임을 알 수 있다. 또한 밝은 회색으로 인해 B성분이 높은 곳에서도 많이 검출 된 것을 확인할 수 있다.

4. 컬러 영상에 대한 histogram equalization을 구현

이론 및 실험예상

이미지의 히스토그램은 특정 밝기 값을 가진 픽셀의 빈도를 나타내는 그래프다. X축은 밝기 값, Y축은 해당 밝기 값을 가진 픽셀 수를 의미한다.

히스토그램 평활화 과정은 원본 이미지의 히스토그램에서 특정 밝기 값에 집중되어있는 경우 각 밝기 값까지의 누적 빈도를 계산하여 밝기 값을 균일하게 분포되도록 만드는 과정이다. 즉 이미지의 명암비를 향상시키기 위해 사용되는 기법이다.

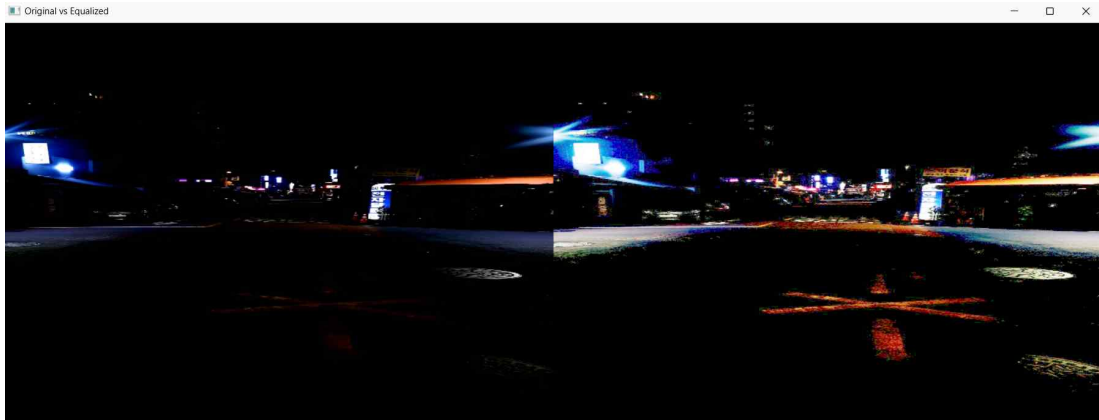
BGR요소를 직접 수정하는 방식과 YCrCb에서 밝기인 Y만을 평활화 하는 두가지 실험을 진행하려한다.

BGR요소를 직접 평활화 할 경우 명암비 뿐만 아니라 색상을 강화할 수 있어 이미지가 더 강한 색상 대비를 가질 수 있다. 하지만 이전 실험에서 알아봤듯이 R,G,B 성분들의 합성으로 Y나 Mg등의 색상이 표현되므로 다른 색상을 표현하는 RGB간의 패턴을 살리지 못하는 문제가 발생해 색상 왜곡의 문제가 발생 할 수 있다.

YCrCb의 경우는 밝기 만을 equalization을 진행하기 때문에 명암비 만을 수정하고 색상은 유지하게 된다. 이로 인해 부드럽고 자연스러운 결과를 얻을 수 있다.

단점으로는 Y단일 채널만을 히스토그램 평활화를 진행하기 때문에 RGB값에 대해 직접적인 히스토그램 평활화를 진행 할 때 보다 명암비와 대비를 크게 강화하지 못한다.

1.BGR요소에 직접 equalization



<어두운 골목길>

어두운 골목길을 촬영한후 BGR채널을 split한후 각 채널에 히스토그램 평활화를 적용하여 화질 개선을 시도한 결과 사진이다.

해당 영상은 좌측의 원본에 비하여 뒤에 있는 자동차나 도로 바닥의 진행금지 표시 등을 확인 할 수 있다. 바닥의 진행금지 표시는 흰색 페인트로 칠해졌지만 RGB성분을 직접 평활화를 하였을 때 붉은 빛을 띄는 것을 알 수 있다.

즉 모든 성분이 골고루 합성되어 흰색을 보여야 하지만 앞서 말한대로 색상 왜곡이 발생한 것을 알 수 있다.

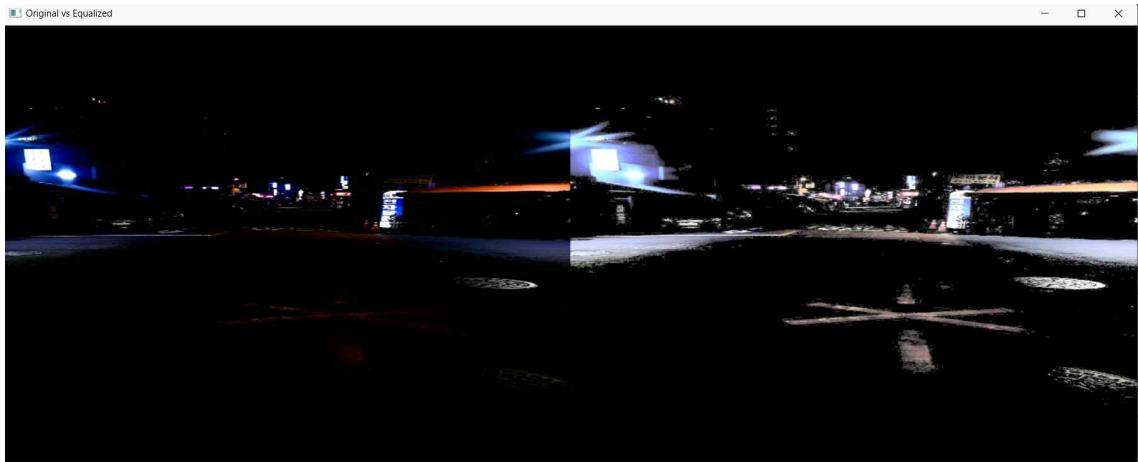


<핸드폰으로 찍은 달사진>

이전에 핸드폰 카메라를 이용하여 찍은 달사진에 대해 화질 개선을 시도하였다.

해당 사진처럼 어두운 배경에서 밝은 물체가 잘 나온 경우 히스토그램 평활화를 진행하면 이미지의 밝기 분포를 고르게 하려하며 밝은 부분이 더 밝아지고 이미지가 더 깨질수 있다. 또한 존재하지 않던 색상이 발생하며 색상 왜곡이 발생한 것을 확인할 수 있다.

2.YCrCb에 대해 equalization



<어두운 골목길>

YCrCb 색상 공간에서 히스토그램 평활화를 통해 화질 개선을 진행하였을 때 직전의 BGR 색상각각에 평활화를 진행하고 병합한 이미지와 달리 Y채널에 해당하는 밝기만을 평활화대상으로 지정하여 색상 왜곡이 발생하지 않았다. 따라서 도로의 진행금지 표시도 흰색 페인트로 칠한 실제 색상을 유지하는 것을 확인할 수 있다. 즉 칼라 영상에서 히스토그램 평활화를 진행할 때 RGB요소를 직접 수정하는 것보다 밝기만을 평활화 적용 대상으로 설정했을 때 색상 왜곡이 발생하지 않아 더 자연스러운 결과를 얻을 수 있음을 확인 할 수 있다.



<핸드폰으로 찍은 달사진>

마찬가지로 핸드폰으로 이전에 찍어두었던 달사진을 평활화를 통해 화질개선을 시도해 보았다. 해당 사진은 특별히 어둡거나 밝은 사진이 아닌 명암대비가 잘되어있는 원본이미지 특성상 평활화를 적용하자 RGB요소를 평활화 했던것과 동일하게 달의 모습도 더 밝아져 이미지가 흐려진 결과를 볼 수 있었다.

다만 밝기인 Y채널만을 평활화 하여 이전의 RGB요소를 평활화 했을 때와 다르게 색상 왜곡은 발생하지 않았다.

결론

화질 개선을 시도할 때 RGB요소 각각을 히스토그램 평활화를 진행한다면 각 요소의 합성으로 표현되는 색들이 왜곡되는 색상 왜곡이 발생할 수 있다.

따라서 YCrCb에서 밝기인 Y채널만을 히스토그램 평활화를 적용하여 원본의 색상을 유지하는 자연스러운 화질개선 방법이 더 좋은 방법이라고 볼 수 있다.

또한 평활화는 명암대비가 한쪽으로 치우친 이미지를 재분배 과정을 거쳐 균등한 히스토그램 분포를 갖도록 하는 알고리즘이다.

이때 앞선 실험에서 사용한 달의 사진과 같이 이미 명암대비가 잘 이루어진 사진을 히스토그램 평활화를 진행한다면 과도한 밝기 증가와 같은 현상이 발생하여 원본 이미지보다 좋지 못한 결과를 얻을 수 있다.