# 과제2 보고서(RGB이미지 합성하여 컬러 이미지 복원하기)

소프트웨어학과 22011831 김형규

원본 이미지(src)는 RGB가 분리된 이미지가 수직으로 붙어있는 이미지이다. 위에서부터 Blue, Green, Red 순서이다. 이를 각각 B이미지, G이미지, R이미지라고 하자.
src를 나누기 위해 B,G,R 이미지의 크기를 먼저 지정해준다. 이미지의 너비는 원본과 같게, 높이는 원본의 1/3 크기로 하였다. src의 너비를 W, 높이를 H라고 하고, B,G,R이미지의 너비를 w, 높이를 h라고 하면, w=W, h=H/3이다.
w*h의 크기로 B, G, R이미지를 생성해준다. 결과 이미지(dst) 역시 똑같은 크기로 생성해준다.
생성한 B,G,R 이미지에 값을 지정해준다. B이미지는 src에서 y좌표가 0~H/3인 부분, G이미지는 원본 이미지에서 y좌표가 H/3~H/3*2인 부분, R이미지는 원본 이미지에서 y좌표가 H/3*2~H인 부분이다.
이중for문을 돌려 바깥 for문은 y가 0~h까지, 안쪽 for문은 x가 0~w까지 증가하도록 돌리고,
B이미지에서의 (x, y)위치의 밝기 값은 src에서 (x, y)위치의 밝기 값으로 저장한다.
G이미지에서의 (x, y)위치의 밝기 값은 src에서 (x, y+H/3)위치의 밝기 값으로 저장한다.
R이미지에서의 (x, y)위치의 밝기 값은 src에서 (x, y+H/3*2)위치의 밝기 값으로 저장한다.
이러면 src를 3등분하여 B, G, R 이미지에 저장할 수 있다. 이제 이 이미지들을 잘 겹쳐지게 정렬하고 dst에 출력하면 컬러 이미지를 복원할 수 있다.

**[가로 경계선, 세로 경계선을 찾는 방법으로 정렬]**
B, G, R 이미지는 모두 검은색 액자처럼 된 틀 안에 그려져 있다는 것을 발견하였다. 그림과 액자 틀의 위쪽 경계선과 왼쪽 경계선을 찾아 그 선을 기준으로 잡고 이미지를 정렬할 수 있겠다는 생각이 들었다. 이미지는 w*h 크기의 2차원 배열로 표현할 수 있고, 각 원소는 0~255의 값을 갖는다. 경계선을 찾을 때는, 이미지의 가운데 행부터 위쪽 방향으로 연산을 진행한다. 이미지의 n번째 행과 n-1번째 행의 밝기 값의 평균을 비교하여 일정 값 이상 차이가 나면 그 행을 경계선으로 구분하는 방식이다.

그림(1)

그림(2)

하지만 이 방법은 틀 안에 있는 그림 자체가 왜곡된 경우에는 사용할 수 없는 방법이다. 그림(1)은 배포된 이미지 중 "pg2.jpg" 파일의 B이미지중의 한 부분이고, 그림(2)는 같은 파일의 R이미지에서 그림(1)과 같은 위치에 있는 부분이다. 그림(1)은 하얀색 사각형이 왼쪽 테두리에 딱 붙어있지만, 그림(2)는 약간 거리가 있는 것을 볼 수 있다. 즉, 검은색 테두리를 기준으로 잡아서 정렬을 하더라도 어긋날 것이다. 이 방법을 통해서는 '각 이미지의 대응되

는 부분을 찾으면 된다.' 라는 아이디어를 얻을 수 있었다.

**[기준 이미지의 기준 행/열과 밝기 값이 차이가 가장 적은 행/열을 찾아서 정렬]**

B이미지의 가운데 행을 기준으로 G이미지와 R이미지에서 대응되는 행을 찾아서 정렬하는 방법이다. 먼저 B이미지의 가운데 행의 밝기 값을 각 픽셀마다 모두 구하고, G이미지와 R이미지에서 그 행과 밝기 값의 차이가 가장 적은 행을 각각 찾는다. 그 행들의 y좌표의 차이가 곧 이미지의 y방향의 오차가 되는 것이다. 가로 방향으로 같은 연산을 적용하여 x방향의 오차를 찾는다.


그림(3)


그림(4)


그림(5)

위의 연산을 B이미지의 중간 행을 기준으로 진행하였다. 그림(3)은 B이미지의 중간에 파란 선을 그은 것이다. 그림(4)와 그림(5)는 위의 연산을 진행하여 컴퓨터가 각각 G이미지와 R이미지에 대응되는 행이라고 판단한 곳에 초록색, 빨간색 선을 그은 것이다. 이렇게 나눠진 이미지는 x방향의 오차와 y방향의 오차가 동시에 있기 때문에 실제로 대응되는 행이더라도 같은 x좌표에서 연산을 진행하면 차이가 커질 수밖에 없는 것이다.

또한, 기준 이미지의 기준 행/열이 이미지의 다른 부분에서도 비슷하게 반복된다면, 정확도가 떨어질 수 있다. 이 방법을 사용하고 난 후, 'x방향의 오차와 y방향의 오차를 동시에 찾아야겠다.'라는 생각이 들었다.

**[기준 영역에 대한 밝기 차이가 가장 적은 곳을 찾아 정렬]**

B이미지의 가운데 영역을 기준으로 해당 영역을 조금씩 옮겨가며 그때마다 R이미지와 G이미지의 같은 위치에서의 밝기 차이가 최소가 될 때를 찾는다. 차이가 최소일 때의 영역을 움직인 만큼의 x좌표와 y좌표를 변수에 저장해두었다가 나중에 dst에 색을 칠할 때 그 좌표만큼 더하여 칠한다. 이 방법을 사용하여 대응되는 부분을 확실하게 찾을 수 있었다. 하지만 이 방법은 연산량이 많다보니 그만큼 시간이 오래 걸린다는 단점이 있었다.

**(시간 줄이기)**

먼저 연산량 자체를 줄이는 방법을 고민해보았다. 연산량을 줄이려면 기준 영역 자체를 줄이는 방법이 있는데, 이 방법을 사용하려면 B, G, R이미지에서의 오차가 원래부터 그리 크지 않아야 한다. 오차를 줄이기 위해, src를 3등분하여 B, G, R이미지에 초기화하기 전에 src의 위쪽과 아래쪽의 흰 공백을 제거한 다음에 3등분을 하면 오차가 줄어들 것이라고 생각했다. 흰 공백을 제거하려면 검은색 테두리가 처음 나오는 위치의 y좌표를 알아야 하는데, 함수를 새로 정의하여 탐색을 진행하였다.

함수 원형 : int findBoundaryVertical(IplImage *src, int startY, int dy);

기능 : 이미지를 인자로 넘겨받아, y좌표가 startY일 때부터 시작하여 dy만큼 증가시키면서,

흰색과 검은색의 경계를 찾는다. dy가 양수일 경우, 위에서부터 아래로 탐색하고, 음수일 경우, 아래에서부터 위로 탐색을 진행한다. y번째 행과 y+dy번째 행의 각 픽셀마다 밝기 차이를 모두 더한 뒤, 너비로 나누면 밝기 차이의 평균을 구할 수 있다. 이 평균의 차이가 어느 정도 크다면, y를 반환한다.
startY에 작은 값, dy는 1로 초기화하면 위쪽 경계선의 y좌표를 찾을 수 있고, 반대로 startY에 src의 높이보다 살짝 작은 값, dy는 -1로 초기화하면 아래쪽 경계선의 y좌표를 찾을 수 있다. 이렇게 해서 찾은 y좌표를 바탕으로 그 사이에 있는 이미지를 3등분하여 B, G, R이미지에 각각 저장한다. 이러면 y방향으로의 오차를 어느 정도 줄일 수 있다.

그리고, 밝기 차이가 가장 적은 영역을 찾는 연산을 진행하기 전에 2차원 배열을 3개 동적 할당 받아서 배열에 각 이미지의 밝기 값들을 저장한다. 이 배열에서 기준 영역에 대한 밝기 차이가 가장 작은 곳을 찾는 연산을 진행하면, 이미지 자체에서 cvGet2D()함수를 이용하여 연산을 진행할 때보다 시간이 적게 걸린다는 것을 알게 되었다.

**[템플릿 매칭]**
이미지 정렬에 대한 정보를 찾던 중, 이러한 이미지 정렬 기술을 '템플릿 매칭'이라고 부른다는 것을 알게 되었다. 보통 인공지능 모델에서 마스크 인식이나, 자동차 인식 등의 분야에 사용하는 기술인데, OpenCV에서도 해당 기술을 수행할 수 있는 함수를 지원한다는 것을 알았다. cvMatchTemplate()함수에 원본이미지와 템플릿 이미지를 인자로 넘겨주면 해당 템플릿이미지를 원본이미지에 조금씩 움직여가며 대조시키고, 정확도가 가장 높은 곳의 좌표를 알 수 있다.
함수 원형 : void cvMatchTemplate(IplImage *src, IplImage *templ, IplImage *result, int method)
기능:src에서 templ이미지를 조금씩 움직여가면서 대조시킨다. 인자로 넘긴 연산방법으로 연산을 진행하여 해당 위치에 대한 정확도를 result에 저장한다.
그 다음 결과 값이 저장된 result이미지를 cvMinMaxLoc()함수를 이용하면 정확도의 최댓값, 그 최댓값의 위치 등을 알 수 있다.
함수 원형 : void cvMinMaxLoc(IplImage *result, double *min, double *max, CvPoint *minPos, CvPoint* maxPos)
기능 : result이미지에 저장된 값 중에서 최댓값과 최솟값을 찾는다. 값을 저장할 변수의 주소값을 인자로 넘기면 그 변수에 값을 저장해준다. 최솟값/최댓값의 좌표 또한 알 수 있다.
B이미지에서 중간 부분을 템플릿이미지로 만들어 G이미지와 R이미지에 템플릿 매칭을 수행하면 대응되는 부분의 좌표를 알 수 있다. 이 좌표와 기준 템플릿의 좌표의 차이가 해당 이미지의 오차가 되는 것이다.