컬러영역에서 관심영역추출

True color Img

- 크기를 구할 수 있어야함

RGB로 딸기영역 추출하기

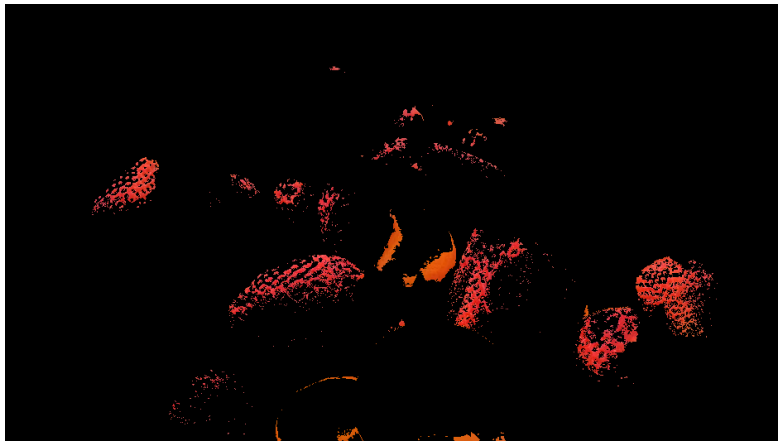
```
829     //딸기 추출하기
830     //BGR 순서인데 Red에 해당하는 값 즉 R이 200보다 큰 값만 내보냄 (마스킹함) -> 나머지는 가린다는 이야기
831 for (int i = 0; i < H; i++) {
832     for (int j = 0; j < W; j++) {
833         //단순하게 딸기니까 빨간색이 많겠지? 라고 생각
834         if (Image[i * W *3 + j*3+2]  > 200) {
835             Output[i * W * 3 + j * 3] = Image[i * W * 3 + j * 3];
836             Output[i * W * 3 + j * 3 + 1] = Image[i * W * 3 + j * 3 + 1];
837             Output[i * W * 3 + j * 3 + 2] = Image[i * W * 3 + j * 3 + 2];
838         }
839         else
840             //전부 0 => black
841             Output[i * W * 3 + j * 3] = Output[i * W * 3 + j * 3 + 1] = Output[i * W * 3 + j * 3 + 2] = 0;
842     }
843 }
844     SaveBMPFile(hf, hInfo, hRGB, Output, hInfo.biWidth, hInfo.biHeight, "output.bmp");
845
```

RGB에서 R이 높다는 건 빨간 값도 많다고 볼 순 있지만
그냥 밝기값이 높을때도 R이 크다

실패! 다른 것도 R이 큰 게 많아
마스킹을 패싱한 것도 많아

심지어 딸기는 다 표시되지도 않았어
기준점을 넘지 못함

```
829     //딸기 추출하기
830     //BGR 순서인데 Red에 해당하는 값 즉 R이 200보다 큰 값만 내보냄 (마스킹함) -> 나머지는 가린다는 이야기
831 for (int i = 0; i < H; i++) {
832     for (int j = 0; j < W; j++) {
833         //단순하게 딸기니까 빨간색이 많겠지? 라고 생각
834         //그럼 R은 밝기도 되니까 G,B도 조정해보자
835         if (Image[i * W *3 + j*3+2]  > 200 && Image[i * W * 3 + j * 3 + 0] < 100 && Image[i * W * 3 + j * 3 + 1] < 100)
836         {
837             Output[i * W * 3 + j * 3] = Image[i * W * 3 + j * 3];
838             Output[i * W * 3 + j * 3 + 1] = Image[i * W * 3 + j * 3 + 1];
839             Output[i * W * 3 + j * 3 + 2] = Image[i * W * 3 + j * 3 + 2];
840         }
841         else
842             //전부 0 => black
843             Output[i * W * 3 + j * 3] = Output[i * W * 3 + j * 3 + 1] = Output[i * W * 3 + j * 3 + 2] = 0;
844     }
845 }
```



그래도 딸기가 아닌 값은 많이 out 시킬 수 있었음

```
828
829     //딸기 추출하기
830     //BGR 순서인데 Red에 해당하는 값 즉 ROI 200보다 큰 값만 내보냄 (마스킹함) -> 나머지는 가린다는 이야기
831 for (int i = 0; i < H; i++) {
832     for (int j = 0; j < W; j++) {
833         //단순하게 딸기니까 빨간색이 많겠지? 라고 생각
834         //그럼 R은 밝기도 되니까 G,B도 조정해보자
835         //근데 글도 ROI 많이 높아서 다른 값을 조정해서 보면 좀 덜 보임 ==> 전통적인 방법 ; 조건을 변경하여 확인해봄
836         if (Image[i * W *3 + j*3+2]  > 130 && Image[i * W * 3 + j * 3 + 0] < 50 && Image[i * W * 3 + j * 3 + 1] < 100)
837         {
838             Output[i * W * 3 + j * 3] = Image[i * W * 3 + j * 3];
839             Output[i * W * 3 + j * 3 + 1] = Image[i * W * 3 + j * 3 + 1];
840             Output[i * W * 3 + j * 3 + 2] = Image[i * W * 3 + j * 3 + 2];
841         }
842         else
843             //전부 0 => black
844             Output[i * W * 3 + j * 3] = Output[i * W * 3 + j * 3 + 1] = Output[i * W * 3 + j * 3 + 2] = 0;
845     }
846 }
```

딸기 부분 좀 더 추가

Green색 감지를 젤 잘 하니까 사람 눈 센싱을 고려하여 Y에서 G를 높게 뽑음

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.16874 & -0.3313 & 0.500 \\ 0.500 & -0.4187 & -0.0813 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} \quad \text{- (식 2-1)}$$

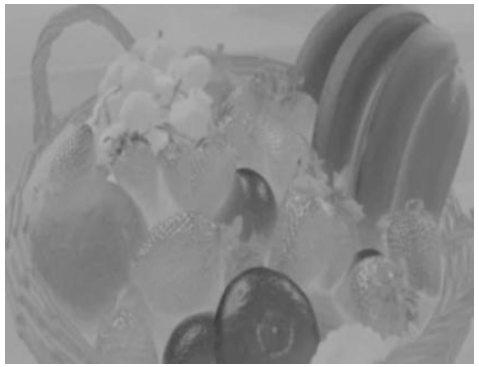
0~255값으로 맞추기 위해 Cb Cr은 +128 해주기

```
682 void RGB2YCbCr(BYTE* Image, BYTE* Y, BYTE* Cb, BYTE* Cr, int W, int H)
683 {
684     for (int i = 0; i < H; i++)
685     {
686         for (int j = 0; j < W; j++)
687         {
688             //GrayScale을 담고있던 y!
689             //이때 주의할 점은, 뒤에가 더블인데 BYTE에 집어넣어줘야해서 형변환을 (BYTE)로 해줘야해
690             Y[i * W + j] = (BYTE)(0.299 * Image[i * W * 3 + j * 3 + 2] + 0.587 * Image[i * W * 3 + j * 3 + 1] + 0.114 * Image[i * W * 3 + j * 3]);
691             Cb[i*W+j] = (BYTE)(-0.168 * Image[i * W * 3 + j * 3 + 2] + -0.3313 * Image[i * W * 3 + j * 3 + 1] + 0.5 * Image[i * W * 3 + j * 3] + 128.0);
692             Cr[i * W + j] = (BYTE)(0.5 * Image[i * W * 3 + j * 3 + 2] + -0.4187 * Image[i * W * 3 + j * 3 + 1] + -0.0813 * Image[i * W * 3 + j * 3] + 128.0);
693         }
694     }
695 }
696
```

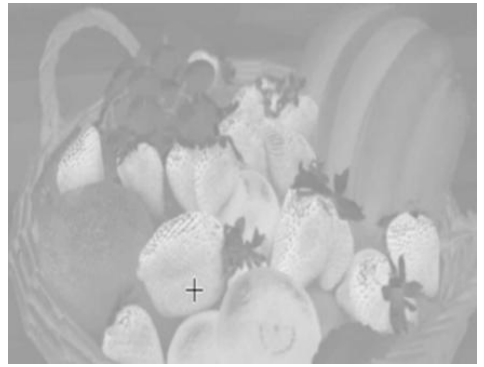
```
818     BYTE* Y = (BYTE*)malloc(ImgSize);
819     BYTE* Cb = (BYTE*)malloc(ImgSize);
820     BYTE* Cr = (BYTE*)malloc(ImgSize);
821
822
823     RGB2YCbCr(Image, Y, Cb, Cr, W, H);
824
825
826     fp = fopen("Y.bmp", "wb");
827     fwrite(Y, sizeof(BYTE), W* H, fp);
828     fclose(fp);
829
830     fp = fopen("Cb.bmp", "wb");
831     fwrite(Cb, sizeof(BYTE), W* H, fp);
832     fclose(fp);
833
834     fp = fopen("Cr.bmp", "wb");
835     fwrite(Cr, sizeof(BYTE), W* H, fp);
836     fclose(fp);
```

Y

Cb



Cr



- 문제 분석
- Cr을 스포이드로 뽑아보면
- Cr값 기준으로 딸기 레드값은 212
- 그런데 귤 값도 200정도가 넘음

- BUT
- Cb부분에서 딸기는 138정도고
- Cb부분에서 귤은 87정도임
- Cb부분에서 딸기는 130보다는 크고 Cr 성분은 200보다 크게 하면 되겠네

- 어두운 부분에선 딸기가 빛을 받으면 R 값도 줄어들어서 RGB 값으로 표현하기 어려워요

- 하지만 이렇게 해도 어려워
- 그래서 딥러닝을 많이 써
- 딸기에 대한 엄청난 이미지를 가지고 딥하게 학습하면 딸기라는 걸 학습 과정중 딸기의 특징또한 알아서 학습하게 돼
- Cb Cr을 우리가 정하고 있는 이 방식이 사실은 전통적인 거야!

```
879
880  BYTE* Y = (BYTE*)malloc(ImgSize);
881  BYTE* Cb = (BYTE*)malloc(ImgSize);
882  BYTE* Cr = (BYTE*)malloc(ImgSize);
883
884
885  RGB2YCbCr(Image, Y, Cb, Cr, W, H);
886
887
888      for (int i = 0; i < H; i++)
889      {
890          for (int j = 0; j < W; j++)
891          {
892              if (Cb[i * W + j] < 130 && Cr[i * W + j]>200)
893              {
894                  Output[i * W * 3 + j * 3] = Image[i * W * 3 + j * 3];
895                  Output[i * W * 3 + j * 3+1] = Image[i * W * 3 + j * 3+1];
896                  Output[i * W * 3 + j * 3+2] = Image[i * W * 3 + j * 3+2];
897              }
898              //블랙으로 보이도록
899              else
900              {
901                  Output[i * W * 3 + j * 3] = Output[i * W * 3 + j * 3 + 1] = Output[i * W * 3 + j * 3+2] = 0;
902              }
903          }
904      }
905
```

과제 : 사람 얼굴

- 얼굴 영역 검출하여 바운딩박스하여 이미지 출력하기

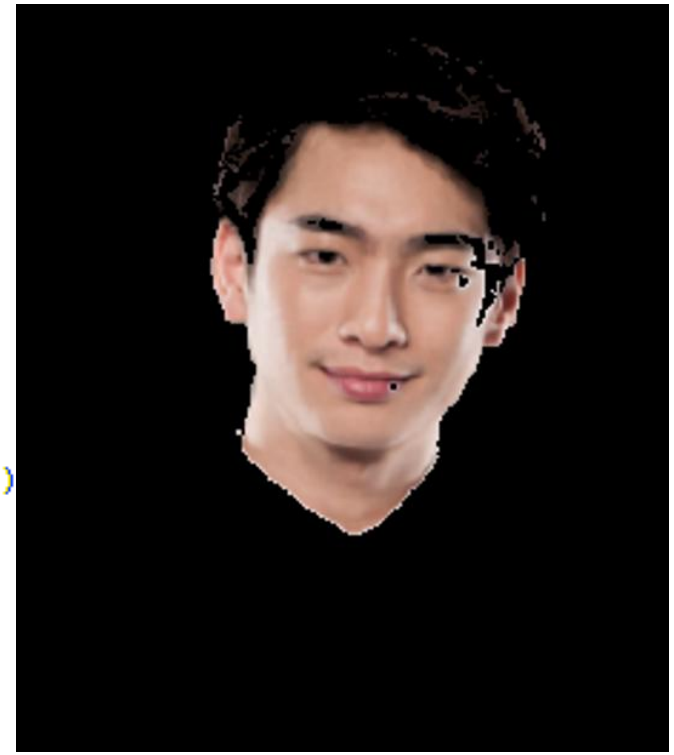
```
879
880     BYTE* Y = (BYTE*)malloc(ImgSize);
881     BYTE* Cb = (BYTE*)malloc(ImgSize);
882     BYTE* Cr = (BYTE*)malloc(ImgSize);
883
884
885     RGB2YCbCr(Image, Y, Cb, Cr, W, H);
886
887     //피부색 영역만 masking
888     for (int i = 0; i < H; i++)
889     {
890         for (int j = 0; j < W; j++)
891         {
892             if ( 125>Cb[i * W + j] >95 && 155< Cr[i * W + j] <185 )
893             {
894                 Output[i * W * 3 + j * 3] = Image[i * W * 3 + j * 3];
895                 Output[i * W * 3 + j * 3 + 1] = Image[i * W * 3 + j * 3 + 1];
896                 Output[i * W * 3 + j * 3 + 2] = Image[i * W * 3 + j * 3 + 2];
897             }
898             //블랙으로 보이도록
899             else
900             {
901                 Output[i * W * 3 + j * 3] = Output[i * W * 3 + j * 3 + 1] = Output[i * W * 3 + j * 3 + 2] = 0;
902             }
903         }
904     }
905
906
907     SaveBMPFile(hf, hInfo, hRGB, Output, hInfo.biWidth, hInfo.biHeight, "output.bmp");
908
909
```

교수님께서 추천해주신 125>Cb>95 && 185>Cr>155 =➔ 결과가 얼굴영역이 포함이 많이 안 되어 있음 =======➔ 이 말은 즉슨! 너무 타이트하다는 뜻임

```
879
880     BYTE* Y = (BYTE*)malloc(ImgSize);
881     BYTE* Cb = (BYTE*)malloc(ImgSize);
882     BYTE* Cr = (BYTE*)malloc(ImgSize);
883
884
885     RGB2YCbCr(Image, Y, Cb, Cr, W, H);
886
887     //피부색 영역만 masking
888     for (int i = 0; i < H; i++)
889     {
890         for (int j = 0; j < W; j++)
891         {
892             if ( 125>Cb[i * W + j] && Cb[i * W + j] >95 && 155< Cr[i * W + j] && Cr[i * W + j] <185 )
893             {
894                 Output[i * W * 3 + j * 3] = Image[i * W * 3 + j * 3];
895                 Output[i * W * 3 + j * 3 + 1] = Image[i * W * 3 + j * 3 + 1];
896                 Output[i * W * 3 + j * 3 + 2] = Image[i * W * 3 + j * 3 + 2];
897             }
898             //블랙으로 보이도록
899             else
900             {
901                 Output[i * W * 3 + j * 3] = Output[i * W * 3 + j * 3 + 1] = Output[i * W * 3 + j * 3 + 2] = 0;
902             }
903         }
904     }
```

피부색이니까 Cr을 점점 늘리면 점점 포용적이게 된다

```
387     //피부색 영역만 masking
388     for (int i = 0; i < H; i++)
389     {
390         for (int j = 0; j < W; j++)
391         {
392             if ( 125>Cb[i * W + j] && Cb[i * W + j] >95 && 130< Cr[i * W + j] && Cr[i * W + j] <220 )
393             {
394                 Output[i * W * 3 + j * 3] = Image[i * W * 3 + j * 3];
395                 Output[i * W * 3 + j * 3 + 1] = Image[i * W * 3 + j * 3 + 1];
396                 Output[i * W * 3 + j * 3 + 2] = Image[i * W * 3 + j * 3 + 2];
397             }
398             //블랙으로 보이도록
399             else
400             {
401                 Output[i * W * 3 + j * 3] = Output[i * W * 3 + j * 3 + 1] = Output[i * W * 3 + j * 3 + 2] = 0;
402             }
403         }
404     }
```



근데 또 머리카락이 거슬려!

```
937     //근데 머리카락 부분이 거슬리지 => 어떻게 하면 좋을까?
938     //라벨링 해서 제일 큰 사이즈 라벨만 남길게!
939
940     //ImgSize의 이진영상이 필요하니까 gray영상 크기로 만들어야해서 3을 나눈다 -> no ImgSize는 지금 W*H라 안 나눠도 돼
941     //이 코드 모르겠다
942     BYTE* temp3 = (BYTE*)malloc(ImgSize);
943     for (int i = 0; i < H; i++)
944     {
945         for (int j = 0; j < W; j++)
946         {
947             if (Output[i * W * 3 + j * 3] == 0)
948             {
949                 temp3[i * W + j] = 0;
950             }
951
952             else
953             {
954                 temp3[i * W + j] = 255;
955             }
956         }
957     }
```

Output값이 현재 Color값이니까 RGB값임
근데 B값이 0인 경우니까 black인 걸 말하는데
그럼 배경 아닌가? =➔ yes!

```
958
959     m_BlobColoring(temp3, H, W); // 라벨링 제일 큰 값만 남기는 것 -> 가장 큰 것만 0으로 남기고 나머지는 255
960     //사각형 그리기
961     int LUX, LUY, RDX, RDY;
962     Obtain2DBoundingBox(temp3, W, H, &LUX, &LUY, &RDX, &RDY);
963     DrawColorRectOutline(Image, W, H, LUX, LUY, RDX, RDY,255, 0, 0);
```

```
// GlassFire 알고리즘을 이용한 라벨링 함수
void m_BlobColoring(BYTE* CutImage, int height, int width)
{
    int i, j, m, n, top, area, Out_Area, index, BlobArea[1000];
    long k;
    short curColor = 0, r, c;
    //  BYTE** CutImage2;
    Out_Area = 1;

    // 스택으로 사용할 메모리 할당
    short* stackx = new short[height * width];
    short* stacky = new short[height * width];
    short* coloring = new short[height * width];

    int arr_size = height * width;

    // 라벨링된 픽셀을 저장하기 위해 메모리 할당

    for (k = 0; k < height * width; k++) coloring[k] = 0;  // 메모리 초기화

    for (i = 0; i < height; i++)
    {
        index = i * width;
        for (j = 0; j < width; j++)
        {
            // 이미 방문한 점이거나 픽셀값이 255가 아니라면 처리 안함
            if (coloring[index + j] != 0 || CutImage[index + j] != 255) continue;
            r = i; c = j; top = 0; area = 1;
            curColor++;

            while (1)
            {
            GRASSFIRE:
                for (m = r - 1; m <= r + 1; m++)
                {
                    index = m * width;
                    for (n = c - 1; n <= c + 1; n++)
                    {
                        //관심 픽셀이 영상경계를 벗어나면 처리 안함
                        if (m < 0 || m >= height || n < 0 || n >= width) continue;

                        if ((int)CutImage[index + n] == 255 && coloring[index + n] == 0)
                        {
                            coloring[index + n] = curColor; // 현재 라벨로 마크
                            if (push(stackx, stacky, arr_size, (short)m, (short)n, &top) == -1) continue;
                            r = m; c = n; area++;
                            goto GRASSFIRE;
                        }
                    }
                }
                if (pop(stackx, stacky, &r, &c, &top) == -1) break;
            }
            if (curColor < 1000) BlobArea[curColor] = area;
        }
    }

    float grayGap = 255.0f / (float)curColor;

    // 가장 면적이 넓은 영역을 찾아내기 위함
    for (i = 1; i <= curColor; i++)
    {
        if (BlobArea[i] >= BlobArea[Out_Area]) Out_Area = i;
    }
    // CutImage 배열 클리어~
    for (k = 0; k < width * height; k++) CutImage[k] = 255;

    // coloring에 저장된 라벨링 결과중 (Out_Area에 저장된) 영역이 가장 큰 것만 CutImage에 저장
    for (k = 0; k < width * height; k++)
    {
        if (coloring[k] == Out_Area) CutImage[k] = 0;  // 가장 큰 것만 저장 (size filtering)
        //if (BlobArea[coloring[k]] > 500) CutImage[k] = 0;  // 특정 면적이상되는 영역만 출력
        //CutImage[k] = (unsigned char)(coloring[k] * grayGap);
    }

    delete[] coloring;
    delete[] stackx;
    delete[] stacky;
}
// 라벨링 후 가장 넓은 영역에 대해서만 뽑아내는 코드 포함
```

```
void Obtain2DBoundingBox(BYTE* Image, int W, int H, int* LUX, int* LUY, int* RDX, int* RDY)
{
    int flag = 0;
    for (int i = 0; i < H; i++) {
        for (int j = 0; j < W; j++) {
            if (Image[i * W + j] == 0) {
                *LUY = i;
                flag = 1;
                break;
            }
        }
        if (flag == 1) break;
    }
    flag = 0;
    for (int i = H - 1; i >= 0; i--) {
        for (int j = 0; j < W; j++) {
            if (Image[i * W + j] == 0) {
                *RDY = i;
                flag = 1;
                break;
            }
        }
        if (flag == 1) break;
    }
    flag = 0;

    for (int j = 0; j < W; j++) {
        for (int i = 0; i < H; i++) {
            if (Image[i * W + j] == 0) {
                *LUX = j;
                flag = 1;
                break;
            }
        }
        if (flag == 1) break;
    }
    flag = 0;
    for (int j = W - 1; j >= 0; j--) {
        for (int i = 0; i < H; i++) {
            if (Image[i * W + j] == 0) {
                *RDX = j;
                flag = 1;
                break;
            }
        }
        if (flag == 1) break;
    }
}
```

```
void DrawColorRectOutline(BYTE* Img, int W, int H, int LU_X, int LU_Y, int RD_X, int RD_Y, BYTE R, BYTE G, BYTE B)

{
    for (int i = LU_X; i < RD_X; i++)
    {
        Img[LU_Y * W *3 + i * 3 + 0] = B;
        Img[LU_Y * W *3 + i * 3 + 1] = G;
        Img[LU_Y * W *3 + i * 3 + 2] = R;
    }
    for (int i = LU_X; i < RD_X; i++)
    {
        Img[RD_Y * W * 3 + i * 3 + 0] = B;
        Img[RD_Y * W * 3 + i * 3 + 1] = G;
        Img[RD_Y * W * 3 + i * 3 + 2] = R;
    }
    for (int i = LU_Y; i < RD_Y; i++)
    {
        Img[i* W * 3 + LU_X * 3 + 0] = B;
        Img[i * W * 3 + LU_X * 3 + 1] = G;
        Img[i * W * 3 + LU_X * 3 + 2] = R;
    }
    for (int i = LU_Y; i < RD_Y; i++)
    {
        Img[i* W * 3 + RD_X * 3 + 0] = B;
        Img[i * W * 3 + RD_X * 3 + 1] = G;
        Img[i * W * 3 + RD_X * 3 + 2] = R;
    }
}
```