EE431 Homework 3

Date: 14.12.2022

In the homework, the color version of the gauss filter is requested to be implemented in this function format:

color **cgaussf(color **img, int flag, int NC, int NR, int count)

For this, some parameters have been added to be used in the function, temp_1 and temp_2 are used for image allocate in NC*NR sizes.

```
color **temp_1;
color **temp_2;
int i , j , k , padding;
temp_1 = (color **)alloc_color_img(NC , NR); // to allocate space for corrected pixel values (same size with img)
temp_2 = (color **)alloc_color_img(NC , NR); // to allocate space for corrected pixel values (same size with img)
padding = count / 2; // define a padding which prevents segmentation faults
```

The flag parameter in this function can take 4 different values, these are 0,1,2 and 3. When the flag is equal to 0, gaussian filtering is applied to R, G and B, which are 3 channels of RGB. As seen in the code block below, gaussing filter is applied to all channels in the innermost for loop. The first loop represents the horizontal mask, the second loop represents the vertical mask.

```
if(flag == 0)
{
    for(i = padding; i < NR - padding; i++)
    {
        for(j = padding; j < NC - padding; j++)
        {
            for(i = 0 ; i < NR ; i++)
            {
                for(j = 0 ; j < NC ; j++)
                {
                    temp_1[i][j].r=img[i][j].r; // to process the gaussian filter for red pixels, and not destroy the original img, temp_1 used as a temporary variable
                    temp_1[i][j].g=img[i][j].g; // to process the gaussian filter for green pixels, and not destroy the original img, temp_1 used as a temporary variable
                    temp_1[i][j].b=img[i][j].b; // to process the gaussian filter for blue pixels, and not destroy the original img, temp_1 used as a temporary variable
                }
            }

            for(k = 0 ; k < count ; k++)
            {
                for(i = 0 ; i < NR ; i++)
                {
                    for(j = 1 ; j < NC - 1 ; j++)
                    {
                        temp_2[i][j].r = (temp_1[i][j-1].r + 2 * temp_1[i][j].r + temp_1[i][j+1].r) / 4; // horizontal mask was done to red pixels and stored in temp_2 variable
                        temp_2[i][j].g = (temp_1[i][j-1].g + 2 * temp_1[i][j].g + temp_1[i][j+1].g) / 4; // horizontal mask was done to green pixels and stored in temp_2 variable
                        temp_2[i][j].b = (temp_1[i][j-1].b + 2 * temp_1[i][j].b + temp_1[i][j+1].b) / 4; // horizontal mask was done to blue pixels and stored in temp_2 variable
                    }
                }
                for(i = 1 ; i < NR - 1 ; i++)
                {
                    for(j = 0 ; j < NC ; j++)
                    {
                        temp_1[i][j].r = (temp_2[i-1][j].r + 2 * temp_2[i][j].r + temp_2[i+1][j].r) / 4; // vertical mask was done to red pixels and stored in temp_1 varible
                        temp_1[i][j].g = (temp_2[i-1][j].g + 2 * temp_2[i][j].g + temp_2[i+1][j].g) / 4; // vertical mask was done to green pixels and stored in temp_1 varible
                        temp_1[i][j].b = (temp_2[i-1][j].b + 2 * temp_2[i][j].b + temp_2[i+1][j].b) / 4; // vertical mask was done to blue pixels and stored in temp_1 varible
                    }
                }
            }
        }
    }
}
```

R channel

When the flag is equal to 1, the gaussian filter is only applied to the R channel, first of all, the original image is destroyed by applying gauss to the red pixels, this image is assigned the temp_1 temporary value. Using the temp_1 parameter, a horizontal mask is applied on the red pixels and this is kept in the temp_2 parameter. Then, it is assigned to the temp_1 variable by applying a vertical mask to the temp_2 variable.

```
else if(flag == 1)
{
    for(i = padding; i < NR - padding; i++)
    {
        for(j = padding; j < NC - padding; j++)
        {
            for(i = 0 ; i < NR ; i++)
            {
                for(j = 0 ; j < NC ; j++)
                {
                    temp_1[i][j].r=img[i][j].r; // to process the gaussian filter for red pixels, and not destroy the original img, temp_1 used as a temporary variable
                }
            }

            for(k = 0 ; k < count ; k++)
            {
                for(i = 0 ; i < NR ; i++)
                {
                    for(j = 1 ; j < NC - 1 ; j++)
                    {
                        temp_2[i][j].r = (temp_1[i][j-1].r + 2 * temp_1[i][j].r + temp_1[i][j+1].r) / 4; // horizontal mask was done to red pixels and stored in temp_2 variable
                    }
                }
                for(i = 1 ; i < NR - 1 ; i++)
                {
                    for(j = 0 ; j < NC ; j++)
                    {
                        temp_1[i][j].r = (temp_2[i-1][j].r + 2 * temp_2[i][j].r + temp_2[i+1][j].r) / 4; // vertical mask was done to red pixels and stored in temp_1 varible
                    }
                }
            }
        }
    }
}
```

G Channel

When the flag is equal to 2, the gaussian filter is only applied to the R channel, first of all, the original image is destroyed by applying gauss to the green pixels, this image is assigned the temp_1 temporary value. Using the temp_1 parameter, a horizontal mask is applied on the green pixels and this is kept in the temp_2 parameter. Then, it is assigned to the temp_1 variable by applying a vertical mask to the temp_2 variable.

```
else if(flag == 2)
{
    for(i = padding; i < NR - padding; i++)
    {
        for(j = padding; j < NC - padding; j++)
        {
            for(i = 0 ; i < NR ; i++)
            {
                for(j = 0 ; j < NC ; j++)
                {
                    temp_1[i][j].g=img[i][j].g; // to process the gaussian filter for green pixels, and not destroy the original img, temp_1 used as a temporary variable
                }
            }

            for(k = 0 ; k < count ; k++)
            {
                for(i = 0 ; i < NR ; i++)
                {
                    for(j = 1 ; j < NC - 1 ; j++)
                    {
                        temp_2[i][j].g = (temp_1[i][j-1].g + 2 * temp_1[i][j].g + temp_1[i][j+1].g) / 4; // horizontal mask was done to green pixels and stored in temp_2 variable
                    }
                }
                for(i = 1 ; i < NR - 1 ; i++)
                {
                    for(j = 0 ; j < NC ; j++)
                    {
                        temp_1[i][j].g = (temp_2[i-1][j].g + 2 * temp_2[i][j].g + temp_2[i+1][j].g) / 4; // vertical mask was done to green pixels and stored in temp_1 varible
                    }
                }
            }
        }
    }
}
```

B channel

When the flag is equal to 3, the gaussian filter is only applied to the R channel, first of all, the original image is destroyed by applying gauss to the blue pixels, this image is assigned the temp_1 temporary value. Using the temp_1 parameter, a horizontal mask is applied on the blue pixels and this is kept in the temp_2 parameter. Then, it is assigned to the temp_1 variable by applying a vertical mask to the temp_2 variable.

```
else if(flag == 3)
{
    for(i = padding; i < NR - padding; i++)
    {
        for(j = padding; j < NC - padding; j++)
        {
            for(i = 0 ; i < NR ; i++)
            {
                for(j = 0 ; j < NC ; j++)
                {
                    temp_1[i][j].b=img[i][j].b; // to process the gaussian filter for blue pixels, and not destroy the original img, temp_1 used as a temporary variable
                }
            }

            for(k = 0 ; k < count ; k++)
            {
                for(i = 0 ; i < NR ; i++)
                {
                    for(j = 1 ; j < NC - 1 ; j++)
                    {
                        temp_2[i][j].b = (temp_1[i][j-1].b + 2 * temp_1[i][j].b + temp_1[i][j+1].b) / 4; // horizontal mask was done to blue pixels and stored in temp_2 variable
                    }
                }
                for(i = 1 ; i < NR - 1 ; i++)
                {
                    for(j = 0 ; j < NC ; j++)
                    {
                        temp_1[i][j].b = (temp_2[i-1][j].b + 2 * temp_2[i][j].b + temp_2[i+1][j].b) / 4; // vertical mask was done to blue pixels and stored in temp_1 varible
                    }
                }
            }
        }
    }
}
```

In the main part, the top left, top right, bottom left, bottom right parts perform the situations when the flag is 0, 1, 2 and 3, respectively.

```
// ---------------------- GAUSSIAN FILTERING MAIN (CALLING THE FUNCTION) PART ----------------------
top_left = cgaussf(img , 0 , NC , NR , count);      // Filtering part: for top-left, count timess gaussian filter was processed, flag 0 situation occured; which includes RGB pixels.
top_right = cgaussf(img , 1 , NC , NR , count);     // Filtering part: for top-right, count times  gaussian filter was processed, flag 1 situation occured; which includes R pixels.
bottom_left = cgaussf(img , 2 , NC , NR , count);       // Filtering part: for bottom-left, count times gaussian filter was processed, flag 2 situation occured; which includes G pixels.
bottom_right = cgaussf(img , 3 , NC , NR , count);      // Filtering part: for bottom-right, count times gaussian filter was processed, flag 3 situation occured; which includes B pixels.

// reconstruct the image by adding up the sliced and filtered pieces according to conditions mentioned below
for(i = 0 ; i < NR ; i++)
{
    for(j = 0 ; j < NC ; j++)
    {
        if(i < NR / 2 && j < NC / 2)
        {
            img_temp[i][j] = top_left[i][j];    // Q1 allocation from top-left array to the img_temp array
        }
        else if(i < NR / 2 && j > NC / 2)
        {
            img_temp[i][j] = top_right[i][j];   // Q2 allocation from top_right array to the img_temp array
        }
        else if(i > NR / 2 && j < NC / 2)
        {
            img_temp[i][j] = bottom_left[i][j]; // Q3 allocation from bottom_left array to the img_temp array
        }
        else if(i > NR / 2 && j > NC / 2)
        {
            img_temp[i][j] = bottom_right[i][j];    // Q4 allocation from bottom_right array to the img_temp array
        }
    }
}
```

To run our code, we used the following commands to compile and execute;

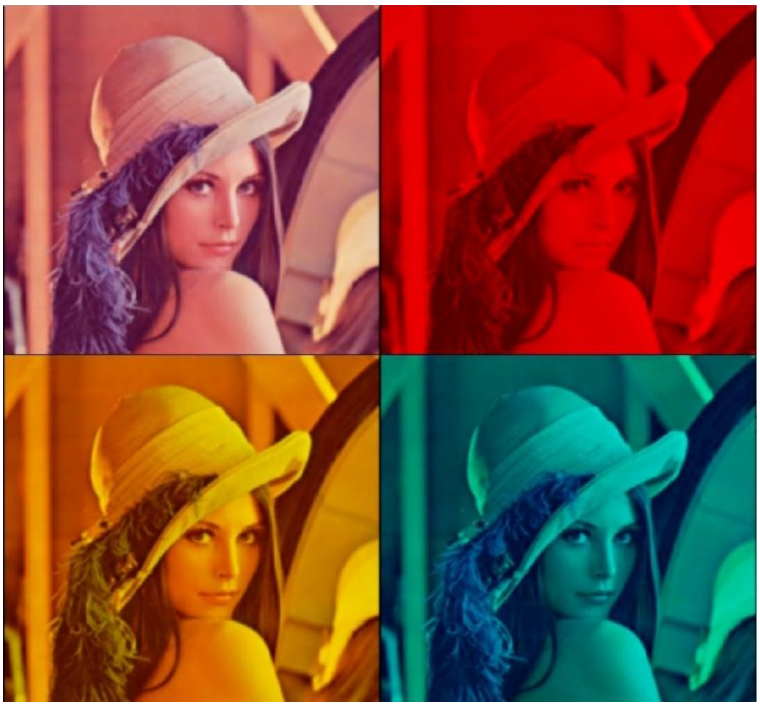
```
gcc hw3.c -o hw3 img_pro.c -lm
```

To see implementation of **1 time** gauss filter

```
. /hw3. lenna.ppm 1
```

To see implementation of **5 time** gauss filter

```
. /hw3. lenna.ppm 5
```



To see implementation of **1 time** gauss filter

```
. /hw3. baboon.ppm 1
```

To see implementation of **5 time** gauss filter

```
. /hw3. baboon.ppm 5
```