WS 2019/20	Exercises Digital Image Processing	Solution Exercise No. 15

Farbbilder

Objective of exercises:

Allgemein

- filter RGB and HSV images
- segmentation of color images

Solution for task 15.1a

Task: *Writes a function filtering an RGB image.*

The filter function is similar to a filter function for gray scale images. The difference is that filtering does not take place via the gray values, but rather separately for each channel. Accordingly, with the mean value filter a separate sum has to be generated for each channel. Figure 1 shows filtering using a 5×5 and a 9×9 mean value filter.

Determination of the interface:

```
// This function filters an RGB image using a mask.
void filter (RgbImage& input, RgbImage& output, GrayImage& mask)
```

Implementation:

```
//
void filter (RgbImage& input, RgbImage& output, GrayImage& mask)
{
    int     mwidth   = mask.getWidth();
    int     mheight  = mask.getHeight();
    float* mdata     = mask.getData();

    int     mwidth2  = mwidth / 2;
    int     mheight2 = mheight / 2;

    int     iwidth   = input.getWidth();
    int     iheight  = input.getHeight();
    Rgb*    idata    = input.getData();
    Rgb*    odata    = output.getData();

    float sum_r, sum_g, sum_b;

    for (int y=0; y<iheight; y++)
    {
        for (int x=0; x<iwidth; x++)
        {
            sum_r = 0.0; sum_g = 0.0; sum_b = 0.0;

            for (int t=0; t<mheight; t++)
            {
                for (int s=0; s<mwidth; s++)
                {
                    int y2 = mod(y+(t-mheight2), iheight);
                    int x2 = mod(x+(s-mwidth2), iwidth);

                    sum_r += mdata[t*mwidth+s] * idata[y2*iwidth+x2].r;
                    sum_g += mdata[t*mwidth+s] * idata[y2*iwidth+x2].g;
                    sum_b += mdata[t*mwidth+s] * idata[y2*iwidth+x2].b;
                }
            }
            odata[y*iwidth+x].r = (unsigned char)sum_r;
            odata[y*iwidth+x].g = (unsigned char)sum_g;
            odata[y*iwidth+x].b = (unsigned char)sum_b;
        }
    }
}

// ------------------------------------------------------------------------
```

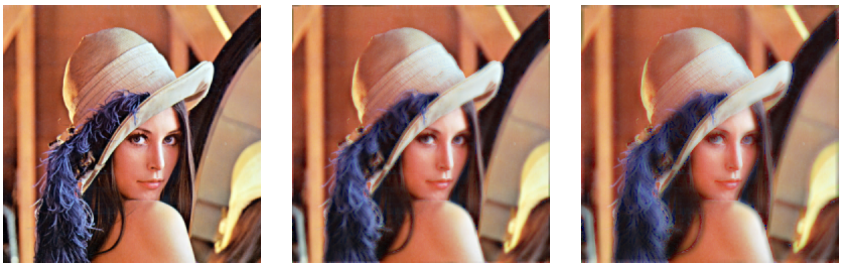
(a) (b) (c)

Figure 1: Input image (a), 5×5 mean value filter (b) and 9×9 mean value filter (c)

Solution for task 15.1b

Task: *Writes a function filtering an HSV image.*

The process for filtering HSV images is different. Only those channels may be filtered that do not contain any color information. Since the color values are given by *Hue* , filtering by color values would cause a change in color. *Saturation* states the color intensity of the color. Only the obscurness/darkness value *Value* does not contain any color information, thus being the only channel that may be filtered. For comparison figur 3 shows filtering of different channels.



(a) (b) (c)

Figure 2: Input image (a), 5×5 mean value filter (b) and 9×9 mean value filter (c)

(a) (b) (c) (d)

Figure 3: Input image (a), 9×9 mean value filteirng of V ((b), corresponds to fig. 2(c)), filtering of S and V (c), filtering of H, S and V (d)

Determination of the interface:

```
// (14.1b) This function filter an HSV image using a mask.
void filter (HsvImage& input, HsvImage& output, GrayImage& mask)
```

Implementation:

```
//
void filter (HsvImage& input, HsvImage& output, GrayImage& mask)
{
    int     mwidth   = mask.getWidth();
    int     mheight  = mask.getHeight();
    float* mdata     = mask.getData();

    int     mwidth2  = mwidth / 2;
    int     mheight2 = mheight / 2;

    int     iwidth   = input.getWidth();
    int     iheight  = input.getHeight();
    Hsv*    idata    = input.getData();
    Hsv*    odata    = output.getData();

    float sum;

    for (int y=0; y<iheight; y++)
    {
        for (int x=0; x<iwidth; x++)
        {
            sum = 0.0;

            for (int t=0; t<mheight; t++)
            {
                for (int s=0; s<mwidth; s++)
                {
                    int y2 = mod(y+(t-mheight2), iheight);
                    int x2 = mod(x+(s-mwidth2), iwidth);

                    sum += mdata[t*mwidth+s] * idata[y2*iwidth+x2].v;
                }
            }

            odata[y*iwidth+x].h = idata[y*iwidth+x].h;
            odata[y*iwidth+x].s = idata[y*iwidth+x].s;
            odata[y*iwidth+x].v = (unsigned char) sum;
        }
    }
}

// ------------------------------------------------------------------------------

//
```

Solution for task 15.2a

Task: *Writes a function to segment the given image segmentation.bmp. Think about an algorithm that with the assistance of recursion compares neighboring pixels and recognizes them based on specific features in order to find connected objects. Emphasize the recognized objects by drawing a rectangle around them.*

In this exercise, the color is the specific feature in the given image. A matrix realized by an object of class `GrayImage` is used to memorize which pixels was already visited. This object stores which pixels have not been visited yet (status 0), which are currently being visited (status 1) and which exhibit the looked-for feature (status 2). At the end only pixels with status 1 and 2 remain in the matrix.
Furthermore in the case of congruence the current position will be saved as propagation/expansion; the propagation/expansion will be adjusted accordingly for additional detected pixels in such a way that in the end a *Bounding Box* remains which is represented by a rectangle (figure 5 (b) and (c)).
From the gray image those segmented areas in which the matrix is scaled can be shown.Tthe matrix contains only values of 1 and 2, which are changed to 0 and 255 by scaling, thus representing the segmented area as white (figure 6 (b) and (c)).

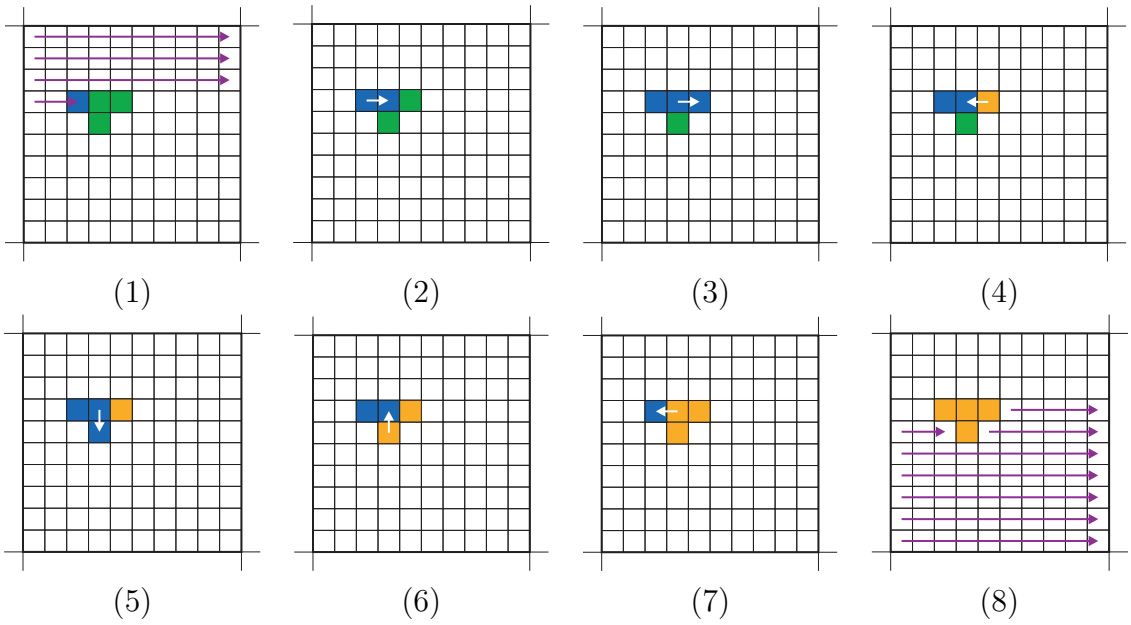


Figure 4: Der Algorithmus bei der Detektion von Objekten, in Einzelschritten

Determination of the interface:

```
// (14.2) Segments the image based on the specified color.
void segmentation (RgbImage& input, RgbImage& output,
                   GrayImage& check, Rgb detectColor, Rgb rectColor)
```

Implementation:

```
//
//
//
void segmentation (RgbImage& input, RgbImage& output,
                   GrayImage& check, Rgb detectColor, Rgb rectColor)
{
    float* cdata  = check.getData();

    int    width  = input.getWidth();
    int    height = input.getHeight();

    int left, top, right, bottom;

    for (int y=0; y<height; y++)
    {
        for (int x=0; x<width; x++)
        {
            left = width-1, top = height-1, right = 0, bottom = 0;


            checkNeighbour (input, check, x, y, left, top, right, bottom, detectColor);

            if (cdata[y*width+x] == 2.0)
            {
                rectangle (output, left, top, right, bottom, rectColor);
            }
        }
    }
}
```

Determination of the interface:

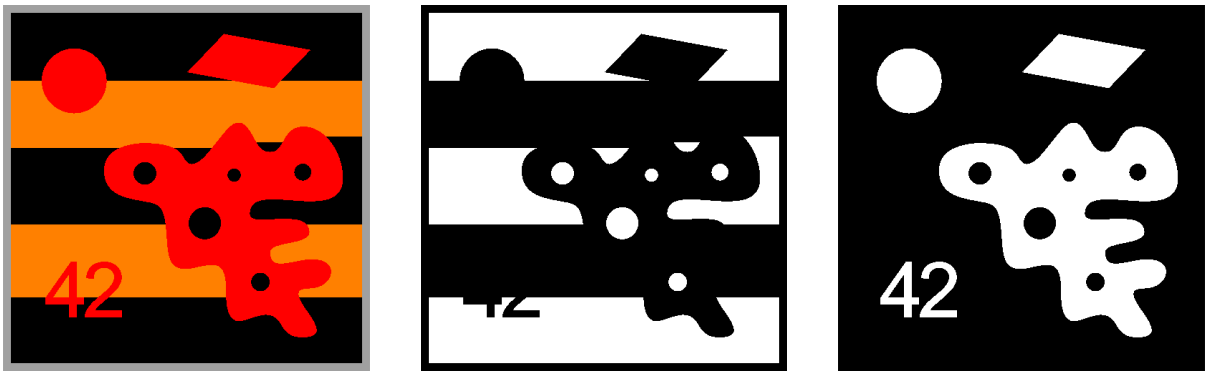
```
// (14.2) Recursive method, examines neighbors and memorizes the
//        expansion coordinates of the object.
void checkNeighbour (RgbImage& input, GrayImage& check, int x, int y,
                     int& x1, int& y1, int& x2, int& y2, Rgb color)
```

Implementation:

```
//
//
//
void checkNeighbour (RgbImage& input, GrayImage& check, int x, int y,
                     int& x1, int& y1, int& x2, int& y2, Rgb color)
{
    int     width  = input.getWidth();
    int     height = input.getHeight();
    int     index  = y*width+x;

    float* cdata  = check.getData();

    if (cdata[index] == 0.0)
    {
        Rgb*   idata = input.getData();

        cdata[index] = 1.0;

        if (idata[index].r == color.r &&
            idata[index].g == color.g &&
            idata[index].b == color.b)
        {
            if (x < x1) x1 = x;
            if (x > x2) x2 = x;
            if (y < y1) y1 = y;
            if (y > y2) y2 = y;

            if (x < width-1)  checkNeighbour (input, check, x+1, y, x1, y1, x2, y2, color);
            if (y < height-1) checkNeighbour (input, check, x, y+1, x1, y1, x2, y2, color);
            if (x > 0)        checkNeighbour (input, check, x-1, y, x1, y1, x2, y2, color);
            if (y > 0)        checkNeighbour (input, check, x, y-1, x1, y1, x2, y2, color);

            cdata[index] = 2.0;
        }
    }
}
```

(a) (b) (c)

Figure 5: Input image (a), bounding box of the segmented areas for black (b) and red (c)



(a) (b) (c)

Figure 6: Input image (a), segmented area represented in white for black (b) and red (c)

Main Program:

```
int main (int argc, char** argv)
{
    /* ********************************** */
    /* Exercise 14                        */
    /* ********************************** */

    Rgb red, blue, green, black;
    red.r   = 255; red.g   =   0; red.b   =   0;
    blue.r  = 128; blue.g  = 128; blue.b  = 255;
    green.r = 0;   green.g = 255; green.b =   0;
    black.r = 0;   black.g =   0; black.b =   0;

    RgbImage input;
    input.load();
    input.show();

    int width  = input.getWidth();
    int height = input.getHeight();

    cout << "Option:\n";
    cout << "1 - RGB und HSV Bild filtern\n";
    cout << "2 - Rote Farbanteile segmentieren\n";
    cout << "3 - Schwarze Farbanteile segmentieren\n";

    int option;
    cin >> option;


    switch (option)
    {
        case 1:
        {
            string filename;
            cout << "Filename der Maske: ";
            cin >> filename;

            RgbImage output1 (width, height);
            RgbImage output2 (width, height);

            HsvImage inputHSV(width, height);
            HsvImage outputHSV(width, height);

            GrayImage mask;
            loadFilterMask(filename, mask);

            filter (input, output1, mask);

            Rgb* idataRGB = input.getData();
            Hsv* idataHSV = inputHSV.getData();
            Rgb* odataRGB = output2.getData();
            Hsv* odataHSV = outputHSV.getData();

            for (int i=0; i<input.getSize(); i++) idataHSV[i] = idataRGB[i];
```

```
            filter (inputHSV, outputHSV, mask);

            for (int i=0; i<input.getSize(); i++) odataRGB[i] = odataHSV[i];

            output1.show();
            output2.show();

            break;
        }
        case 2:
        {
            RgbImage output;
            output.copy(input);

            GrayImage check (width, height);
            check.fill(0);

            segmentation (input, output, check, red, blue);
            output.show();

            scale (check);
            check.show();

            break;
        }
        case 3:
        {
            RgbImage output;
            output.copy(input);

            GrayImage check (width, height);
            check.fill(0);

            segmentation (input, output, check, black, green);
            output.show();

            scale (check);
            check.show();

            break;
        }
        default:
        {
            exit(0);
        }
    }
    cout << "FINISHED.\n";
    return 0;
}
```