3. Exercise – Theory

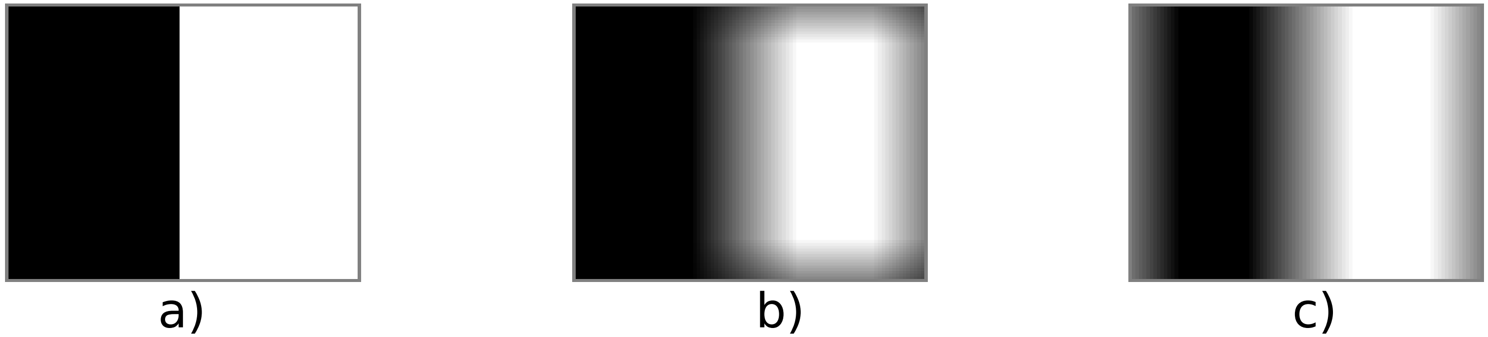


Figure 1

A moving average filter was applied to the image in Figure 1a).

Figure 1b) shows the result, if the convolution is carried out in spatial domain, Figure 1c) if the convolution is carried out as multiplication in frequency domain.

i) Explain which assumptions lead to the “unexpected” border values in each image and why they are different for both methods. *Hint: Consider which are the grey values assumed outside the image boundaries.*

ii) What steps are necessary for the convolution in spatial domain to produce the result in Fig. 1c)? *Hint: Consider different kinds of boundary treatment.*

iii) What steps are necessary for the convolution by multiplication in frequency domain to produce the result in Fig. 1b)? *Hint: Consider to enlarge your image.*

3. Exercise – Given

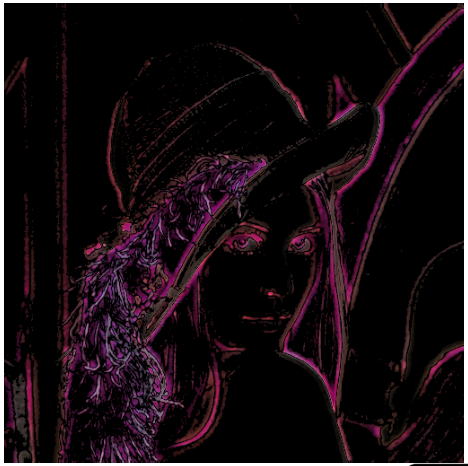
```
FILE: main.cpp
int main(int argc, char** argv)
```

- Declares variables
- Displays and saves images
- Calls Dip3::run-function for unsharp masking, using either:
  - convolution in spatial domain (1)
  - convolution by multiplication in frequency domain (2)

➔ Measures and saves time to files

  ➔ 1: convolutionSpatialDomain.txt

  ➔ 2: convolutionFrequencyDomain.txt



3. Exercise – Given

```
Mat Dip3::mySmooth(Mat& in, int size, int type)
```

`in`: input image

`size`: size of filter kernel

`type`: whether or not working in spatial domain

`return`: smoothed image

- Applies Gaussian blurring to image
- Either working in spatial or frequency domain

```
void Dip3::test(void)
void Dip3::test_createGaussianKernel(void)
void Dip3::test_circShift(void)
void Dip3::test_frequencyConvolution(void)
```

→ Simple test function to check for basic correctness

3. Exercise – Reuse!

```
Mat Dip3::spatialConvolution(Mat& src, Mat& kernel)
```

- Parameter:
  - → `src` : source image
  - → `kernel` : kernel of the convolution
  - → `return` : output image
- Applies convolution in spatial domain
- One method of border handling: size(src) == size(return)
- Do **NOT** use convolution functions of OpenCV

3. Exercise – To Do

Mat Dip3::createGaussianKernel(int kSize)

kSize: the kernel size

return: the generated filter kernel

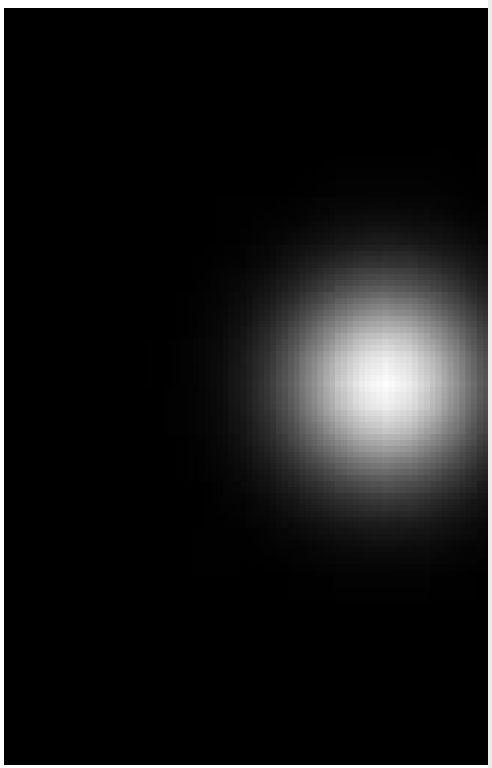
- (

```
Mat Dip3::createGaussianKernel(int kSize){

        Mat kernel = Mat::zeros(kSize, kSize, CV_32FC1);

        // some variables for gaussian filter kernel
        float mu_x = kernel.cols/2;     // x-coordinate of center
        float sigma_x = kernel.cols/5;  // standard deviation in x direction
        float mu_y = kernel.rows/2;     // y-coordinate of center
        float sigma_y = kernel.rows/5;  // standard deviation in y direction

        float val=0, norm=0;

        // calculate corresponding kernel value at each position
        for(float x=0; x<kernel.cols; x++){
            for(float y=0; y<kernel.rows; y++){
                        val = exp( -0.5*(((x-mu_x)/sigma_x)*((x-mu_x)/sigma_x) + ((y-mu_y)/sigma_y)*((y-mu_y)/sigma_y)) );
                        norm += val;
                        kernel.at<float>(y,x) = val;
            }
        }
        // ensure integration to 1
        for(float x=0; x<kernel.cols; x++){
            for(float y=0; y<kernel.rows; y++){
                        kernel.at<float>(y, x) /= norm;
            }
        }
cout << kernel << endl;
        return kernel;

}
```

3. Exercise – To Do

```
Mat Dip3::circShift(Mat& in, int dx, int dy)
```

- Performes circular shift in (dx,dy) direction



```
Mat Dip3::circShift(Mat& in, int dx, int dy){

        Mat out = in.clone();

        int x, y, new_x, new_y;

        for(y=0; y<in.rows; y++){

              // calulate new y-coordinate
              new_y = y + dy;
              if (new_y<0)
                        new_y = new_y + in.rows;
              if (new_y>=in.rows)
                        new_y = new_y - in.rows;

              for(x=0; x<in.cols; x++){

                  // calculate new x-coordinate
                  new_x = x + dx;
                  if (new_x<0)
                        new_x = new_x + in.cols;
                  if (new_x>=in.cols)
                        new_x = new_x - in.cols;

                  out.at<float>(new_y, new_x) = in.at<float>(y, x);
            }
        }
        return out;
}
```

3. Exercise – To Do

`Mat Dip3::frequencyConvolution(Mat& in, Mat& kernel)`

- Calculates convolu
- Forward transform:
  dft(Mat, Mat, 0);
- Inverse transform
  dft(Mat, Mat, DFT_IN
- Spectrum multiplic
  mulSpectrums(Mat, M

```
Mat Dip3::frequencyConvolution(Mat& in, Mat& kernel){
	Mat out = Mat::zeros( in.rows, in.cols, CV_32FC1);

	// create new matrices of optimal size
	Mat dft_in = in.clone();
	Mat dft_kernel = Mat::zeros( in.rows, in.cols, CV_32FC1);
	// DFT
	dft( dft_in, dft_in, CV_DXT_FORWARD, in.rows);

	// copy kernel into new matrix
	Mat roi = dft_kernel(Rect(0,0,kernel.cols,kernel.rows));
	kernel.copyTo(roi);
	// center filterkernel
	dft_kernel = circShift( dft_kernel, -kernel.cols/2, -kernel.rows/2);

	// DFT
	dft( dft_kernel, dft_kernel, CV_DXT_FORWARD );

	// multiplication of fouriertransformed image and kernel
	mulSpectrums( dft_in, dft_kernel, dft_in, 0 );

	// inverse dft
	dft( dft_in, dft_in, CV_DXT_INV_SCALE, out.rows );

	// copy into output image
	dft_in.copyTo(out);

	return out;
}
```

Re Y_{00}	Re
Re Y_{10}	Re
Re Y_{20}	Re
. . .	

3. Exercise – To Do

Mat Dip3::run(Mat& in, int type, int size, double thresh, double scale)
Mat Dip3::usm(Mat& in, int type, int size, double thresh, double scale)

`in`: input image
`type`: smoothing i
by multiplic
`size`: kernel size k
`thresh`: threshold T
`scale`: scale s
`return`: result

- Unsharp maskin
- Thresholding to
  - Useful function

```
      // substract smoothed version from original
      tmp = in - tmp;

      // threshold "edge" image
  Mat tmp1, tmp2;
      threshold(tmp, tmp1, thresh, 0, THRESH_TOZERO);
      threshold(-tmp, tmp2, thresh, 0, THRESH_TOZERO);
  tmp = tmp1-tmp2;

      // scale edge enhancement
      tmp *= scale;

      // add "edge enhancement" image
      Mat out = in + tmp;

      // constrain image values to be in [0,255]
      threshold(out, out, 0, 0, THRESH_TOZERO);
      threshold(out, out, 255, 255, THRESH_TRUNC);

      return out;
```