Labwork 5: Gausian Blur Convolution

Nguyen Thanh Binh

October 10, 2025

1 Gaussian blur filter:

To create the filter, we need the size of the filter, which is 7×7. To get the filter, each pixel is computed by using the normal distribution. μ_x, μ_y are the means of the x and y-axis, which is the mean of the length of the filter.

If even:

$$\mu_x = \frac{H}{2}$$

If odd:

$$\mu_x = \frac{H-1}{2}$$

For μ_y, replace H with W.

The gaussian filter formula is shown in the following:

$$G(x,y) = \frac{1}{2\pi\sigma^2} \exp\left[-\frac{(x-\mu_x)^2 + (y-\mu_y)^2}{2\sigma^2}\right]$$

Below here is the code implementation:

```
    import math
@cuda.jit
def gaussian_blur(layer, stdev):
  tidx = cuda.threadIdx.x
  tidy = cuda.threadIdx.y
  meanx = layer.shape[0] // 2
  meany = layer.shape[1] // 2

  layer[tidx,tidy] = ( 1/( 2*math.pi*(stdev**2)))*
  math.exp(-( (tidx-meanx)**2 + (tidy-meany)**2) / ( 2*(stdev**2) ))
```

`layer` argument is the kernel, and `stdev` is the standard deviation

2 Convolution:

2.1 Without shared memory

The convolution sum is calculated using the formula below, the size of the kernel is 7×7, then we will have the formula below:

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{17} \\ x_{21} & x_{22} & \cdots & x_{27} \\ \vdots & \vdots & \ddots & \vdots \\ x_{71} & x_{72} & \cdots & x_{77} \end{bmatrix} \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{17} \\ y_{21} & y_{22} & \cdots & y_{27} \\ \vdots & \vdots & \ddots & \vdots \\ y_{71} & y_{72} & \cdots & y_{77} \end{bmatrix}$$

$$= \sum_{i=1}^{7} \sum_{j=1}^{7} x_{(7-i)(7-j)} y_{(1+y)(1+j)}$$

Below here is the code implementation:

```
  @cuda.jit
def convolve(img, layer, res):
  tidx = cuda.threadIdx.x + cuda.blockIdx.x * cuda.blockDim.x
  tidy = cuda.threadIdx.y + cuda.blockIdx.y * cuda.blockDim.y
  # Boundary checking: if the thread coordinates are outside of the image, ignore the threa
  if (i >= tidx) or (j >= tidy):
        return
  delta_tidx = layer.shape[0] // 2
  delta_tidy = layer.shape[1] // 2

  conv_sum = 0
  for i in range (layer.shape[0]):
    for j in range(layer.shape[1]):
      in_x = tidx - i + delta_tidx
      in_y = tidy - j + delta_tidy
      if (in_x >=0) and (in_x < img.shape[0]) and (in_y >=0) and (in_y < img.shape[1]):
        conv_sum += layer[i,j] * img[in_x,in_y]
  res[tidx,tidy] = conv_sum
```

2.2 With shared memory

```
  @cuda.jit
def shared_blur(input,output,kernel):
  tile_image = cuda.shared.array(shape = (7,7), dtype = np.uint8)

  tidx = cuda.threadIdx.x + cuda.blockIdx.x * cuda.blockDim.x
  tidy = cuda.threadIdx.y + cuda.blockIdx.y * cuda.blockDim.y
  tile_image[cuda.threadIdx.x, cuda.threadIdx.y] = input[tidx,tidy]
  cuda.syncthreads()

  if (tidx >= output.shape[0]) or (tidy >= output.shape[1]):
    return

  delta_tidx = kernel.shape[0] // 2
  delta_tidy = kernel.shape[1] // 2

  conv_sum = 0
  for i in range (kernel.shape[0]):
    for j in range(kernel.shape[1]):
      in_x = tidx - i + delta_tidx
      in_y = tidy - j + delta_tidy
      if (in_x >=0) and (in_x < output.shape[0]) and (in_y >=0) and (in_y < output.shape[1]
        conv_sum += kernel[i,j] * input[in_x,in_y]
  output[tidx,tidy] = conv_sum
```

Here are the comparisons before gaussian blurring, and after:

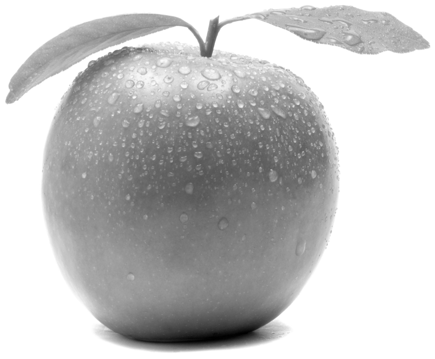

Figure 1: Image before blurring

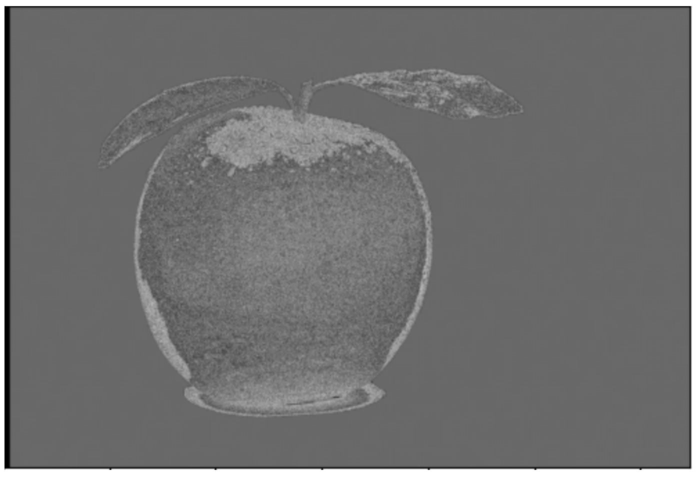

Figure 2: Image after blurring