BV2

Nik Brouw, Tim Stolp

April 2019

1 Theory

In all cases the border is handled by repeating the signal.

1.1

f = {1 $\underline{2}$ 1}
g = {0 0 0 0 $\underline{1}$ 1 1 1 1}
$f \star g$ = {1 0 0 1 3 4 4 4 3}

1.2

f = {0 0 0 0 $\underline{1}$ 1 1 1 1}
g = {1 $\underline{2}$ 1}
$f \star g$ ={6 7 7}

1.3

f = {0 0 0 0 $\underline{1}$ 1 1 1 1}
g = {-1 1}
$f \star g$ ={-1 1}

1.4

$$(f * g)(x) = \sum_{y \in E} f(y)g(x-y)$$

Let $p = x - y$, so $y = x - p$ and $p \in E$

$$\begin{aligned}(f * g)(x) &= \sum_{p \in E} f(x-p)g(p) \\ &= \sum_{p \in E} g(p)f(x-p)\end{aligned}$$

This is the convolution of $(g * f)(x)$ so $(f * g)(x) = (g * f)(x)$

1.5

$$(f * g)(x) = \sum_{y \in E} f(y)g(x-y)$$

Associativity proof:

$$\begin{aligned}((f * g) * h)(x) &= \sum_{y \in E} (f * g)(y)h(x-y) \\ &= \sum_{y \in E} \sum_{p \in E} f(p)g(y-p)h(x-y) \\ &= \sum_{p \in E} \sum_{y \in E} f(p)g(y)h(x-y-p) \\ &= \sum_{y \in E} f(p)(g * h)(x-p) \\ &= (f * (g * h))(x)\end{aligned}$$

So $((f * g) * h)(x) = (f * (g * h))(x)$

1.6

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & \underline{1} & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

1.7

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & \underline{3} & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

1.8

$$\begin{bmatrix} 0 & 0 & 0 & \underline{0} \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

1.9

Rotating using convolution is impossible because a rotation is not a translation-invariant operation. For example, first rotating and then translating gives a different result as when you first translate and then rotate.

1.10

$$\frac{1}{9} * \begin{bmatrix} 1 & 1 & 1 \\ 1 & \underline{1} & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

1.11

Getting the median of a 3x3 neighbourhood using convolution is impossible because getting the median is not a linear operation. The median is the middle number in a sorted series of numbers. The 3x3 neighbourhood is not necessarily sorted, and thus the median can occur on every place in the neighbourhood making it impossible to use convolution to get the median.

1.12

Getting the minimum value of a 5x5 neighbourhood using convolution is impossible because getting the minimum is not a linear operation. The minimum value can occur on every place in the neighbourhood making it impossible to use convolution to get the minimum.

1.13

$$\frac{1}{5} * \begin{bmatrix} \underline{1} & 1 & 1 & 1 & 1 \end{bmatrix}$$

1.14

$$\frac{1}{16} * \begin{bmatrix} 1 & 2 & 1 \\ 2 & \underline{4} & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

1.15

Take image $\mathbf{M}$ then the unsharp mask is:

$$\mathbf{M} - \frac{1}{16} * \begin{bmatrix} 1 & 2 & 1 \\ 2 & \underline{4} & 2 \\ 1 & 2 & 1 \end{bmatrix} \star \mathbf{M}$$

1.16

$$\frac{1}{2} * \begin{bmatrix} 1 & \underline{0} & -1 \end{bmatrix}$$

1.17

$$\frac{1}{4} * \begin{bmatrix} 1 & \underline{-2} & 1 \end{bmatrix}$$

1.18

Not possible because in order to zoom in on a picture you need to do different kinds of operations on different pixels.

1.19

In order to do thresholding multiple different inputs would need to have the same output. So doing this with a kernel is not possible as it is not a linear operation.

1.20

Since the Gaussian contains an exponential function, it is a smooth function. From the definition of a smooth function follows that the Gaussian has derivatives of all orders everywhere in its domain, thus the derivatives are in arbitrary order.

1.21

Taking the derivative is a translation invariant linear operator, thus it can be written as a convolution.

1.22

$\partial(f \star G_\sigma) = D \star (f \star G_\sigma) = f \star (D \star G_\sigma) = f \star (\partial G_\sigma)$
This formula explains that the derivative of an image at scale σ can be computed by convolution with the derivative of a Gaussian kernel of that scale.
Both associativity and commutativity are used in the step from $D \star (f \star G_\sigma)$ to $f \star (D \star G_\sigma)$.

1.23

Since the Gaussian kernel is an exponential function, taking the partial derivative of it will only multiply the Gaussian by the partial derivative of the power, as shown below:

$$\begin{aligned}\frac{\partial G_\sigma}{\partial x}(x) &= G_\sigma(x) * \frac{\partial}{\partial x}(-\frac{x^2}{2\sigma^2}) \\ &= -\frac{\frac{\partial}{\partial x}(x^2)}{2\sigma^2} * G_\sigma(x) \\ &= -\frac{x}{\sigma^2} * G_\sigma(x)\end{aligned} \tag{1}$$

1.24

The second partial derivative of the Gaussian will be the partial derivative of the first partial derivative, given above.

$$\begin{aligned}\frac{\partial^2 G_\sigma}{\partial x^2}(x) &= \frac{\partial}{\partial x}(-\frac{x}{\sigma^2} * G_\sigma(x)) \\ &= (\frac{\partial(-\frac{x}{\sigma^2})}{\partial x} * G_\sigma(x)) * (\frac{\partial G_\sigma}{\partial x}(x) * (-\frac{x}{\sigma^2})) \\ &= (-\frac{1}{\sigma^2} * G_\sigma(x)) * (\frac{x^2}{\sigma^4} * G_\sigma(x)) \\ &= (\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}) * G_\sigma(x)\end{aligned} \tag{2}$$

1.25

let f be a Gaussian kernel and g be an image.

Smoothing of a Gaussian smoothed image
$f \star (f \star g)$
Can be rewritten as
$(f \star f) \star g$
With f being the Gaussian kernel $\frac{1}{(\sqrt{2\pi\sigma})^d} e^{-\frac{x^2}{2\sigma^2}}$, d being its dimension, multiplying it by itself gives a new Gaussian kernel h with a bigger dimension: $\frac{1}{(\sqrt{2\pi\sigma})^{(d+d)}} e^{-\frac{2x^2}{2\sigma^2}}$. Then, since h is a Gaussian kernel, $h \star g$ will also be a Gaussian smoothing.
$\sigma_{12} = \sqrt{\sigma_1^2 + \sigma_2^2}$

1.26

We can smooth in 2D using:

$$\begin{aligned}G\sigma_{12}(x,y) &= \frac{1}{(\sqrt{2\pi}\sigma_{12})^2} e^{-\frac{x^2+y^2}{2\sigma_{12}^2}} \\ &= \frac{1}{2\pi\sigma_{12}^2} e^{-\frac{x^2+y^2}{2\sigma_{12}^2}} \\ &= \frac{1}{\sqrt{2\pi}\sigma_1} e^{-\frac{x^2}{2\sigma_1^2}} \cdot \frac{1}{\sqrt{2\pi}\sigma_2} e^{-\frac{y^2}{2\sigma 2^2}} \\ &= G\sigma_1(x) \cdot G\sigma_2(y)\end{aligned} \tag{3}$$

Comparing this to the 1D smoothing, $G\sigma(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2}{2\sigma^2}}$, it is visible that the 2D smoothing is made of two 1D smoothing actions.
Thus, the 2D Gaussian is separable.
The scale of $G\sigma_{12}(x,y) = \sqrt{\sigma_1^2 + \sigma_2^2}$.

1.27

As seen in 1.24, taking the second derivative is the same as taking the derivative over the derivative. From this, we can create a basic understanding of what happens when deriving multiple times. The derivative will always be of the form $f * G_\sigma$, with f being a series of σ-functions, e.g. $(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2})$. Every further derivative will multiply f by the derivative of $G_\sigma, (-\frac{x}{\sigma^2})$, and by the derivative of f itself.

1 Matlab

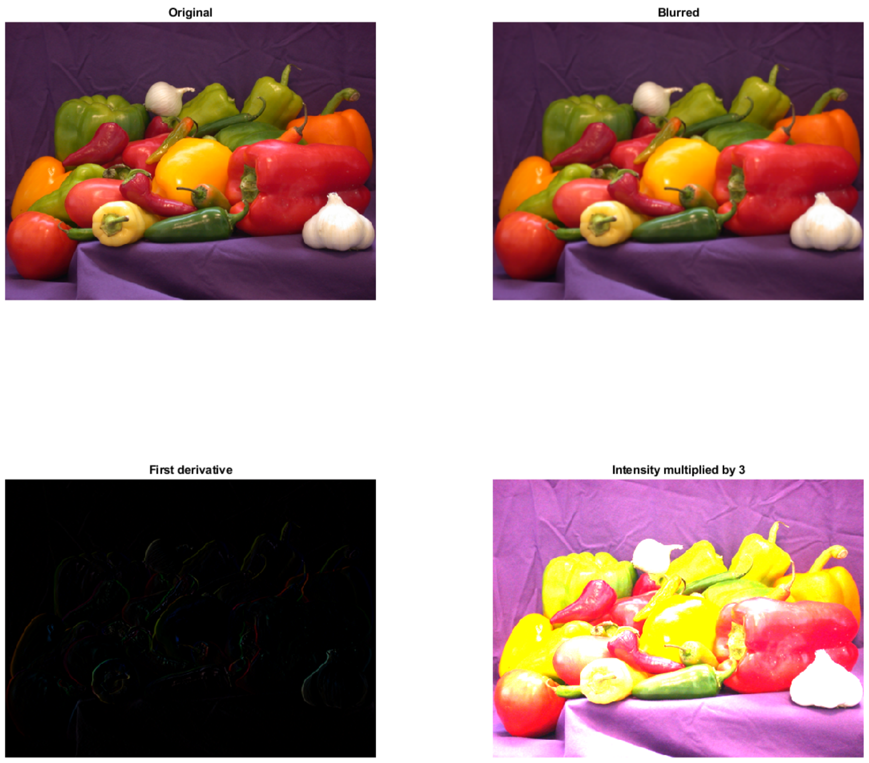

Figure 1: Image with different filters

2 Matlab

We have chosen M and N to be 3*sigma. This number is chosen as it generally fully captures the whole Gaussian curve of that sigma without adding too many values that would have nearly no effect on the result. This seems to be a good balance between speed and accuracy.

Summing all the values together of the Gaussian kernel you would expect it to be 1. This is because when you apply it to a uniform image it should not change any values. Matlab's calculations do not result in a sum of 1, but below 1. To avoid this problem we divide every number in the kernel by the sum of the kernel. This makes the sum of the kernel always 1.

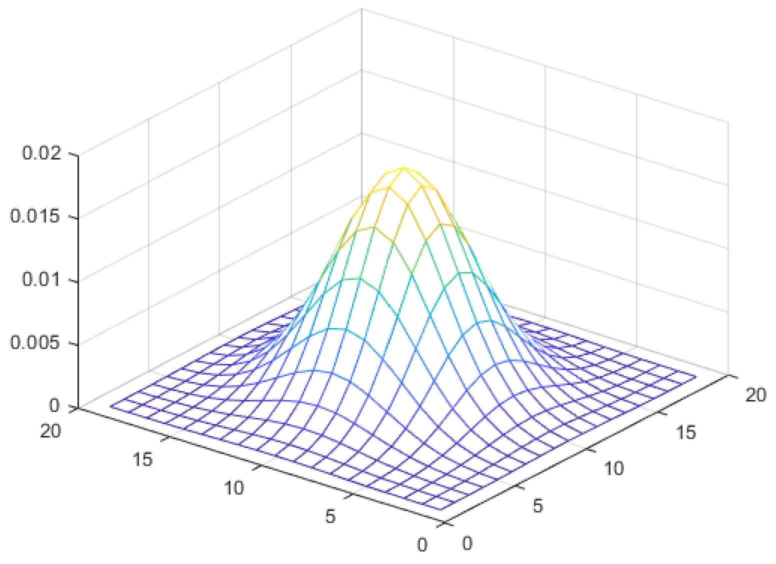

Figure 2: Mesh plot of Gaussian kernel with scale 3

The physical unit of σ is pixels.

To measure the computation time a convolution was done on the 'peppers.png' image with a Gaussian kernel for scales one to two hundred. For each scale the average of 10 computations is taken. The order of computational complexity is exponential and thus O(n^2).

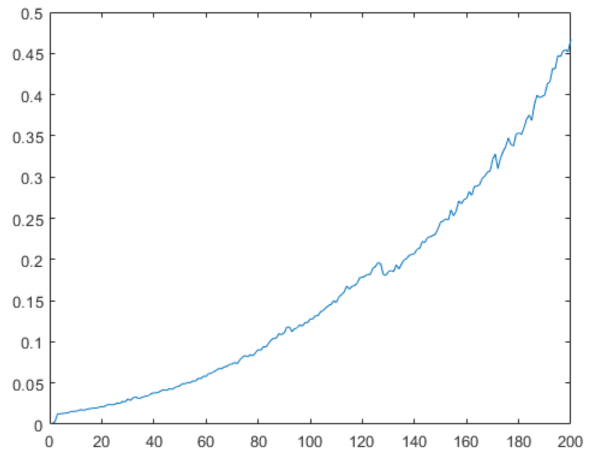

Figure 3: Computation time of convolution with 2D Gaussian kernel plotted against scale

Blurring a 384x512x3 dimensional image two times with a scale of 3 and 4 and blurring an image with scale of 5 gives us a squared error of 20422. Compared to the total amount of pixels, 589824, this is a relatively small amount. This error is likely caused by not fully capturing the whole Gaussian kernel as the kernel dimensions are chosen to be 6*sigma+1 as a balance between accuracy and computation time.

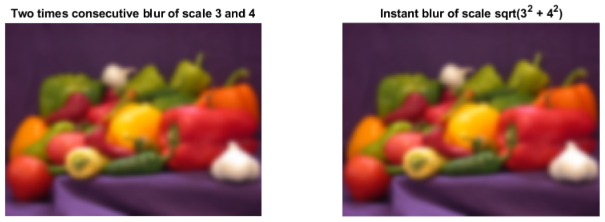
Two times consecutive blur of scale 3 and 4
Instant blur of scale sqrt(3^2 + 4^2)

Figure 4: Comparison of consecutive and one time blur

Using two separated 1d kernels instead of one 2d kernel results in much lower computation time and is clearly linearly correlated with the scale. The computational complexity is O(n).

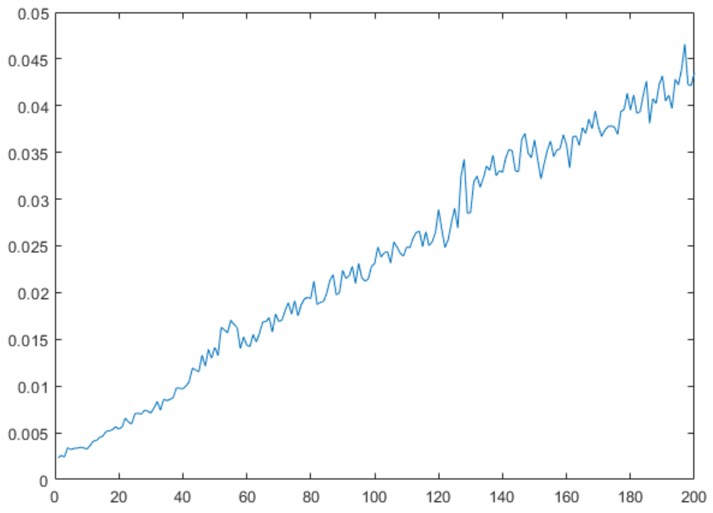

Figure 5: Computation time of convolution with two 1D Gaussian kernels plotted against scale

```
function output = gD(f, sigma, xorder, yorder)
X = Gauss1(sigma);
Y = Gauss1(sigma);

scale = 3*sigma;

x = -scale : scale ;
% Get polynomials for derivative order one and two.
firstpoly = x.*(-1/sigma^2);
secondpoly = ((x.^2).*(sigma^4)) - (1/sigma^2);

% Multiply the Gaussian kernel with the right polynomial
% to get the right Gaussian derivative kernel
if xorder == 1
    X = X.*firstpoly;
end
if xorder == 2
    X = X.*secondpoly;
end
if yorder == 1
    Y = Y.*firstpoly;
end
if yorder == 2
    Y = Y.*secondpoly;
end

% Convolute image with X and Y kernels.
output = imfilter(f, X, 'conv', 'replicate');
output = imfilter(output, Y', 'conv', 'replicate');
end
```

This function was used to create the following visual representation of the 2-jet of f.

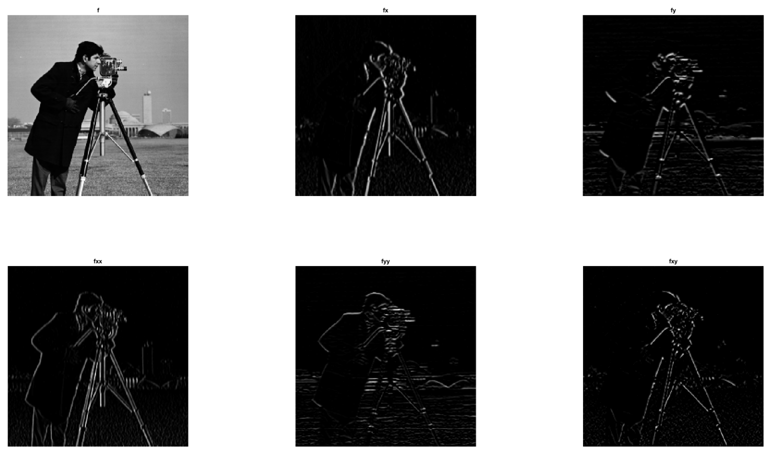

Figure 6: 2-Jet of f

3

We can determine the partial derivatives from the derivatives of cosine and sine. The variables x and y are split over both parts, meaning a part will be constant when not deriving over that variable. This leads to the following derivatives:

1. $f_x = A * V * cos(Vx)$
2. $f_y = -B * W * sin(Wy)$
3. $f_{xx} = -A * V^2 * sin(Vx)$
4. $f_{yy} = -B * W^2 * cos(Wy)$
5. $f_{xy} = 0$

Plots of the original function, the first derivative to x, and the first derivative to y.

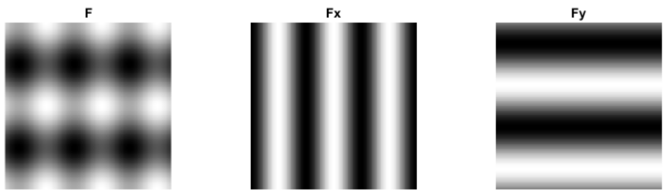

Figure 7: f, f_x, f_y

Plot of original function with gradient vectors.

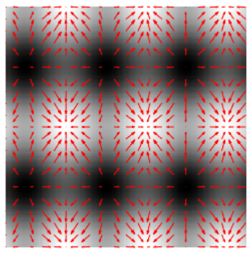

Figure 8: f with gradient vectors

Plot of original function rotated 10 degrees with gradient vectors.

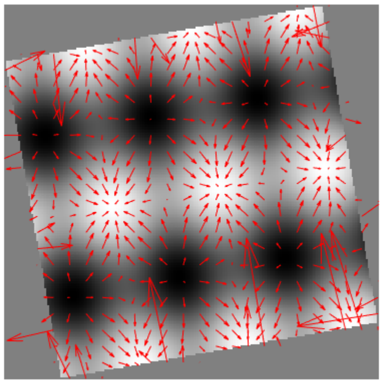

Figure 9: f rotated 10 degrees with gradient vectors

$$f_w = \sqrt{f_x^2 + f_y^2} \tag{4}$$

$$f_{ww} = \frac{1}{f_x^2 + f_y^2}(f_x^2 f_{xx} + 2 f_x f_y f_{xy} + f_y^2 f_{yy}) \tag{5}$$

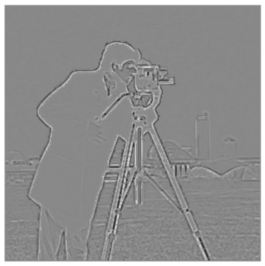

Figure 10: Edges by canny edge detector